

CS 6v81 - Network Security

TCP/IP Security - 1

(Source: Vigna slides & papers)

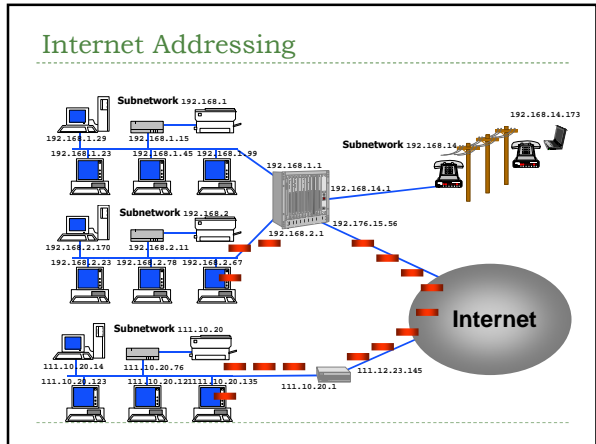
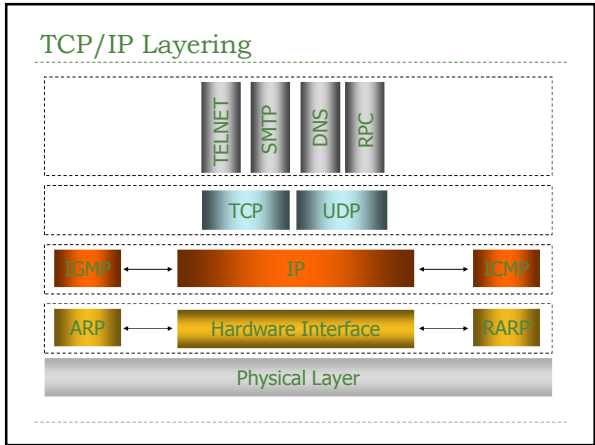
TCP/IP security

- ▶ TCP/IP protocol security
- ▶ Real-time communication security
- ▶ IPsec
- ▶ SSL/TLS
- ▶ BGP security
- ▶ Firewalls and IDSeS
- ▶ Wireless security
- ▶ Anonymous routing

2

The TCP/IP Protocol Suite

- ▶ Network protocols (OSI layer 3)
 - ▶ IP (*Internet Protocol*)
 - ▶ ICMP (*Internet Control Message Protocol*)
 - ▶ IGMP (*Internet Group Management Protocol*)
- ▶ Transport protocols (OSI layer 4)
 - ▶ TCP (*Transfer Control Protocol*)
 - ▶ UDP (*User Datagram Protocol*)
- ▶ Application protocols (OSI layer 7)
 - ▶ SMTP, FTP, SSH, ...
- ▶ Other protocols (OSI layer 2)
 - ▶ ARP (*Address Resolution Protocol*)
 - ▶ RARP (*Reverse Address Resolution Protocol*)



IP Addresses

- ▶ Each host has one or more IP addresses *for each network interface*
- ▶ IP addresses are composed of 32 bit (class+netid+hostid)
- ▶ Represented in dotted-decimal notation: 129.110.11.25
- ▶ Classes
 - ▶ *Class A* (0): netid=7 bit (128 networks, actually 1-126), hostid=24 bit (16777216 hosts)
 - ▶ *Class B* (10): netid=14 bit (16384 networks), hostid=16 bit (65536 hosts)
 - ▶ *Class C* (110): netid=21 bit (2097152 networks), hostid=8 bit (256 hosts)
 - ▶ *Class D - Multicast* (1110): multicast addresses
 - ▶ *Class E* (1111): reserved or future use

Special Addresses

- ▶ As source and destination address
 - ▶ Loopback interface: 127.X.X.X (usually 127.0.0.1)
- ▶ As source address
 - ▶ netid=0, hostid=0 or hostid=XXX: this host on this net (used in special cases such as booting procedures)
- ▶ As destination address
 - ▶ All bits set to 1: local broadcast
 - ▶ netid + hostid with all bits set to 1: net-directed broadcast to netid
- ▶ Reserved addresses (RFC 1597):
 - ▶ 10.0.0.0 - 10.255.255.255
 - ▶ 172.16.0.0 - 172.31.255.255
 - ▶ 192.168.0.0 - 192.168.255.255

Classless Inter-Domain Routing (CIDR)

- ▶ Allocation of large chunks of IP addresses wasted an enormous number of IP addresses
- ▶ Number of hosts is increasing
- ▶ IPv6 provides a larger address space but it is not ubiquitously implemented
- ▶ CIDR is a new addressing scheme for the Internet which allows for more efficient allocation of IP addresses than the old "Class A, B, and C" address scheme
- ▶ The netid/hostid boundary can be placed on any bit between 13 and 27
 - ▶ 32 hosts minimum
 - ▶ 524,288 hosts maximum

Internet Protocol (IP)

- ▶ The IP protocol represents the "glue" of the Internet
- ▶ The IP protocol provides a connectionless, unreliable, best-effort datagram delivery service (delivery, integrity, ordering, non-duplication, and bandwidth is not guaranteed)
- ▶ IP datagrams can be exchanged between any two nodes (provided they both have an IP address)
- ▶ For direct communication IP relies on a number of different lower-level protocols, e.g., Ethernet, Token Ring, FDDI, RS-232

IP Datagram

0	4	8	12	16	20	24	28	31
Version	HL	Service type (TOS)		Total length				
Identifier				Flags		Fragment offset		
Time To Live		Protocol		Header checksum				
Source IP address								
Destination IP address								
Options						Padding		
Data								

IP Header

- ▶ Normal size: 20 bytes
- ▶ Version (4 bits): current value=4 (IPv4)
- ▶ Header length (4 bits): number of 32-bit words in the header, including options (max header size is 60 bytes)
- ▶ Type of service (8 bits): priority (3 bits), quality of service (4 bits), and an unused bit
- ▶ Total length (16 bits): datagram length in bytes (max size is 65535 bytes)
- ▶ Id (16 bits): unique identifier for the datagram (usually incremented by one)

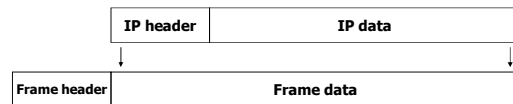
IP Header

- ▶ Flags (3 bits) and offset (13 bits): used for fragmentation
- ▶ Time To Live (8 bits): specifies the max number of hops in the delivery process
- ▶ Protocol (8 bits): specifies the protocol encapsulated in the datagram data (e.g., TCP)
- ▶ Header checksum (16 bits): checksum calculated over the IP header
- ▶ Addresses (32+32 bits): IP addresses of the source and destination of the datagram

IP Options

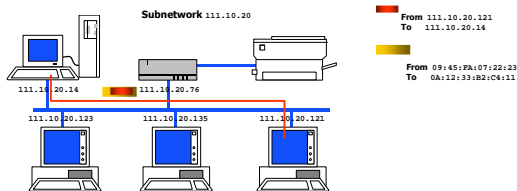
- ▶ Variable length
- ▶ Identified by first byte
 - ▶ *Security and handling restrictions*: used in military applications
 - ▶ *Record route*: each router records its IP address
 - ▶ *Time stamp*: each router records its IP address and time
 - ▶ *Source route*: specifies a list of IP addresses that must be traversed by the datagram
 - ▶ *Loose*: some
 - ▶ *Strict*: all of them

IP Encapsulation



IP: Direct Delivery

- ▶ If two hosts are in the same physical network the IP datagram is encapsulated in a lower level protocol and delivered directly



Ethernet

- ▶ Widely-used link-layer protocol
- ▶ Uses CSMA/CD (Carrier Sense, Multiple Access with Collision Detection)
- ▶ Destination address: 48 bits (e.g., 09:45:FA:07:22:23)
- ▶ Source address: 48 bits
- ▶ Type: 2 bytes (IP, ARP, RARP)
- ▶ Data:
 - ▶ Min 46 bytes (padding may be needed)
 - ▶ Max 1500 bytes
- ▶ CRC: Cyclic Redundancy Check, 4 bytes

Ethernet Frame

dest (6) | src (6) | type (2) | data (46-1500) | CRC (4)

0800 | IP datagram

0806 | ARP (28) | PAD (18)

0808 | RARP (28) | PAD (18)

Address Resolution Protocol (ARP)

- ▶ The address resolution protocol allows a host to map IP addresses to the link-level addresses associated with the peer's hardware interface (e.g., Ethernet) to be used in direct delivery
- ▶ ARP messages are encapsulated in the underlying link level protocol

Address Resolution Protocol

- ▶ Host A wants to know the hardware address associated with IP address I_b of host B
- ▶ A broadcasts a special message to all the hosts on the same physical link
- ▶ Host B answers with a message containing its own link-level address
- ▶ A keeps the answer in its cache (20 minutes)
 - ▶ Can be checked with `arp -a`
- ▶ To optimize traffic, when A sends its request, A includes its own IP address
- ▶ The receiver of the ARP request will cache the requester mapping

ARP Messages

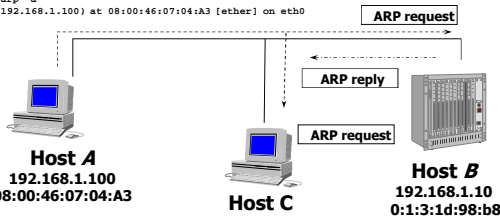
Hw type	Prot type	Hw size	Prot size	Op	Sender Ether	Sender IP	Target Ether	Target IP
---------	-----------	---------	-----------	----	--------------	-----------	--------------	-----------

- ▶ Hardware (2 bytes), protocol (2 bytes), hardware size (1 byte), and protocol size (1 byte) specify the link and network addresses to be mapped (usually Ethernet and IP, respectively) [0x0001, 0x0800, 6, 4]
- ▶ OP field specifies if this is an ARP request or an ARP reply (1= ARP req, 2=ARP reply)
- ▶ Sender Ethernet/IP: data of the requester
- ▶ Target Ethernet: empty in a request
- ▶ Target IP: requested IP address

ARP Request

```

hosta# arp -a
hosta# ping 192.168.1.10
8:0:46:7:4:a3 ff:ff:ff:ff:ff:ff arp 60: arp who-has 192.168.1.10 tell 192.168.1.100
0:1:3:1d:98:b8 8:0:46:7:4:a3 arp 60: arp reply 192.168.1.10 is-at 0:1:3:1d:98:b8
8:0:46:7:4:a3 0:1:3:1d:98:b8 ip 98: 192.168.1.100 > 192.168.1.10: icmp: echo request
0:1:3:1d:98:b8 8:0:46:7:4:a3 ip 98: 192.168.1.10 > 192.168.1.100: icmp: echo reply
hosta# arp -a
hostb (192.168.1.10) at 00:01:03:1d:98:b8 [ether] on eth0
hostb# arp -a
hosta (192.168.1.100) at 08:00:46:07:04:A3 [ether] on eth0
    
```



Proxy ARP and Gratuitous ARP

- ▶ Proxy ARP
 - ▶ Allows a host to answer to ARP requests in place of the actual host
 - ▶ Used by routers to register for traffic to be forwarded between hosts within the same subnet address but on different links
- ▶ Gratuitous ARP
 - ▶ Request sent by a host at bootstrap time requesting its own mapping... no answer expected!
 - ▶ Used to determine if the address is already in use
 - ▶ Used to update the cache entries of other hosts on the same links (e.g., in case of a change of hardware address)

Reverse Address Resolution Protocol

Hw type	Prot type	Hw size	Prot size	Op	Sender Ether	Sender IP	Target Ether	Target IP
---------	-----------	---------	-----------	----	--------------	-----------	--------------	-----------

- ▶ Message format as in ARP (OP: 3=RARP request, 4=RARP reply)
- ▶ RARP is used by diskless stations to obtain their own IP address during startup
- ▶ The client host broadcasts a request
- ▶ All the RARP servers on the same link answer providing the IP address associated with the hardware interface of the client

Local Area Network Attacks

Local Area Network Attacks

- ▶ Sniffing
- ▶ Spoofing/Hijacking
- ▶ ARP attacks
- ▶ RARP attacks
- ▶ Goals
 - ▶ Impersonation of a host
 - ▶ Denial of service
 - ▶ Access to information
 - ▶ Tamper with delivery mechanisms

Network Sniffing

- ▶ Technique at the basis of many attacks
- ▶ The attacker sets his/her network interface in *promiscuous mode*
- ▶ Can access all the traffic on the segment



Why Sniffing?

- ▶ Many protocols (TELNET, FTP, POP, HTTP) transfer authentication information *in the clear*
- ▶ By sniffing the traffic it is possible to collect usernames/passwords, files, mail, etc.
- ▶ Many tools available

Dsniff

- ▶ dsniff at <http://www.monkey.org/~dugsong/dsniff/>
- ▶ Collection of tools for network auditing and penetration testing
- ▶ dsniff, filesnarf, mailsnarf, msgsnarf, urlsnarf, and webspdy passively monitor a network for interesting data (passwords, e-mail, files, etc.)
- ▶ arpspoof, dnsspoof, and macof facilitate the interception of network traffic normally unavailable to an attacker (e.g. due to layer-2 switching)
- ▶ sshmitm and webmitm implement active man-in-the-middle attacks against redirected SSH and HTTPS

Detecting Sniffers on Your System

- ▶ Sniffers are typically passive programs
- ▶ They put the network interface in *promiscuous mode* and listen for traffic
- ▶ They can be detected by programs such as:
 - ▶ ifconfig

```
eth0      Link encap:Ethernet  HWaddr 00:10:4B:E2:F6:4C
          inet addr:192.168.1.20  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:1016 errors:0 dropped:0 overruns:0 frame:0
          TX packets:209 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
```
 - ▶ cpm (*Check Promiscuous Mode*)
 - ▶ ifstatus

Detecting Sniffers on Your Network

- ▶ Suspicious DNS lookups
 - ▶ Sniffer attempts to resolve names associated with IP addresses (may be part of normal operation)
 - ▶ Trap: generate connection from fake IP address not in local network and detect attempt to resolve name
- ▶ Latency
 - ▶ Assumption: Since the NIC is in promiscuous mode EVERY packet is processed
 - ▶ Use ping to analyze response time of host A
 - ▶ Generate huge amount of traffic to other hosts and analyze response time of host A

Detecting Sniffers on Your Network

- ▶ Kernel behavior
 - ▶ Linux
 - ▶ When in promiscuous mode, some kernels will accept a packet that has the wrong Ethernet address but the right destination IP address
 - ▶ Windows 95, 98, NT
 - ▶ When in promiscuous mode, only the first octet is checked for Ethernet broadcast addresses (ff:00:00:00:00:00 will be accepted)
- ▶ AntiSniff tool (<http://www.packetstormsecurity.nl/sniffers/antisniff>)
 - ▶ Covers the techniques above
 - ▶ Uses TCP SYN and TCP handshake forged traffic to overload sniffer when testing latency

TCPDump: Understanding the Network

- ▶ TCPDump is a tool that analyzes the traffic on a network segment
- ▶ One of the most used/most useful tools
- ▶ Based on libpcap, which provides a platform-independent library and API to perform traffic sniffing
- ▶ TCPDump and libpcap are both available at <http://www.tcpdump.org>
- ▶ Allows one to specify an expression that defines which packets have to be printed
- ▶ Requires root privileges to be able to set the interface in promiscuous mode (privileges not needed when reading from file)

TCPDump: Command Line Options

- ▶ -e: print link-level addresses
- ▶ -n: do not translate IP addresses to FQDN names
- ▶ -x: print each packet in hex
- ▶ -i: use a particular network interface
- ▶ -r: read packets from a file
- ▶ -w: write packets to a file
- ▶ -s: specify the amount of data to be sniffed for each packet (e.g., set to 65535 to get the entire IP packet)
- ▶ -f: specify a file containing the filter expression

TCPDump: Filter Expression

- ▶ A filter expression consists of one or more *primitives*
- ▶ Primitives are composed of a *qualifier* and an *id*
- ▶ Qualifiers
 - ▶ type: defines the kind of entity
 - ▶ host (e.g., "host longboard", where "longboard" is the id)
 - ▶ net (e.g., "net 129.110")
 - ▶ port (e.g., "port 23")
 - ▶ dir: specifies the direction of traffic
 - ▶ src (e.g., "src host longboard")
 - ▶ dst
 - ▶ src and dst

TCPDump: Filter Expression

- ▶ Qualifiers (continued)
 - ▶ proto: specifies a protocol of interest
 - ▶ ether (e.g., "ether src host 00:65:FB:A6:11:15")
 - ▶ ip (e.g., "ip dst net 192.168.1")
 - ▶ arp (e.g., "arp")
 - ▶ rarp (e.g., "rarp src host 192.168.1.100")
- ▶ Operators can be used to create complex filter expression
 - ▶ and, or, not (e.g., "host shortboard and not port ftp")
- ▶ Special keywords
 - ▶ gateway: checks if a packet used a host as a gateway
 - ▶ less and greater: used to check the size of a packet
 - ▶ broadcast: used to check if a packet is a broadcast packet

TCPDump: Filter Expression

- ▶ Other operators
 - ▶ Relational: <, >, >=, <=, =, !=
 - ▶ Binary: +, -, *, /, &, |
 - ▶ Logical: and, or, not
 - ▶ "not host longboard and dst host 192.168.1.1"
- ▶ Access to packet data
 - ▶ proto [expr : size] where expr is the byte offset and size is an optional indicator of the number of bytes if interest (1, 2, or 4)
 - ▶ ip[0] & 0xf != 5 to filter only IP datagrams with options

TCPDump: Examples

```
# tcpdump -i eth0 -n -x
# tcpdump -s 65535 -w traffic.dump src host hitchcock
% tcpdump -r traffic.dump arp
# tcpdump arp[7] = 1
# tcpdump gateway csgw and \ ( port 21 or port 20 \)
```

Libpcap

- ▶ `pcap_lookupdev`
 - ▶ looks up a device
- ▶ `pcap_open_live`
 - ▶ opens a device and returns a handle
- ▶ `pcap_open_offline` and `pcap_dump_open`
 - ▶ read from and save packets to files
- ▶ `pcap_compile` and `pcap_setfilter`
 - ▶ set a tcpdump-like filter
- ▶ `pcap_loop`
 - ▶ register a callback to be invoked for each received packet

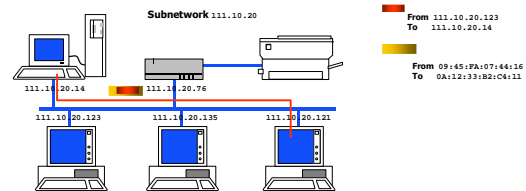
Packet Structure

- ▶ Header is returned in structure

```
struct pcap_pkthdr {
    struct timeval ts; /* time stamp */
    bpf_u_int32 caplen; /* length of portion */
    bpf_u_int32 len; /* length this packet (off wire) */
};
```
- ▶ The actual packet is returned as a pointer to memory
- ▶ Packet can be parsed by “casting” with proto-specific structs
- ▶ Whenever you deal with packets remember that endianness is important
 - ▶ Use `ntohs`, `htons`, `ntohl`, `htonl`
- ▶ Nice tutorial at <http://tcpdump.org/pcap.htm>

IP Spoofing

- ▶ A host impersonates another host by sending a datagram with the address of some other host as the source address
- ▶ The attacker sniffs the network looking for replies from the attacked host



Why IP Spoofing?

- ▶ IP spoofing is used to impersonate sources of security-critical information (e.g., a DNS server or a NIS server)
- ▶ IP spoofing is used to exploit address-based authentication
 - ▶ RPC
 - ▶ “r”-commands (`rsh`, `rcp`, etc.)
- ▶ Many tools available
 - ▶ Protocol-specific spoofers (DNS spoofers, NFS spoofers, etc)
 - ▶ Generic IP spoofing tools

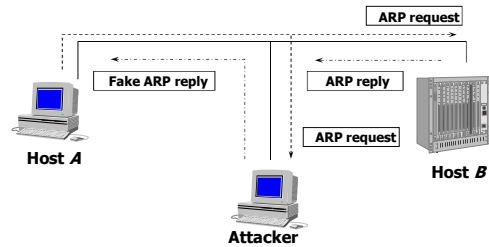
Hijacking

- ▶ Sniffing/Spoofing is the base for hijacking
- ▶ The attacker waits for a client request
- ▶ Races against legitimate host when producing a reply
- ▶ We will see ARP-, UDP-, and TCP-based variations of this attack

Attacks to ARP

- ▶ ARP does not provide any means of authentication
- ▶ Racing against the queried host it is possible to provide a false IP address/link-level address mapping
- ▶ Fake ARP queries/responses can be used to store wrong ARP mappings in a host cache
- ▶ In both cases, the net effect is the redirection of traffic to the attacker (at least for the lifetime of the cache entry)
- ▶ Used in denial-of-service and spoofing attacks

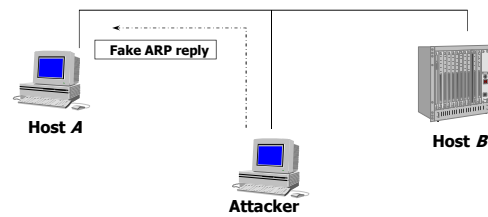
ARP Attack



ARP Attack

- ▶ Since ARP is “stateless” it is possible to provide a fake reply even if a request has not been sent

ARP Attack



Libnet

- ▶ Available at <http://www.packetfactory.net/Projects/Libnet>
- ▶ Provides a platform-independent library of functions to build (and inject) arbitrary packets
- ▶ Allows to write Ethernet spoofed frames
- ▶ Steps in building a packet
 1. Memory Initialization (allocates memory for packets)
 2. Network Initialization (initializes the network interface)
 3. Packet Construction (fill in the different protocol headers/payloads)
 4. Packet Checksums (compute the necessary checksums - some of them could be automatically computed by the kernel)
 5. Packet Injection (send the packet on the wire)

ARP Attack

- ▶ ARP can be used to perform complete traffic redirection
- ▶ Plain ARP spoofing is used against two hosts A and B
- ▶ ARP messages are sent continuously to keep cache updated with the “wrong” information

ARP Attack

- ▶ Attacker creates two alias interfaces with A's and B's IP addresses
- ▶ Attacker's interfaces ARP functions are disabled with `ifconfig -arp`
- ▶ Attacker's interfaces ARP caches are set to the correct values using `arp -s host hw_addr`
- ▶ Attacker sets IP forwarding between the two interfaces

ARP Attack

- ▶ Variation on the previous attack: use ARP to impersonate the gateway and filter all the traffic to external networks
- ▶ Variation: use ARP to map gateway IP to non-existent MAC address (denial-of-service)
- ▶ Gratuitous ARP: spoofed messages can be used to broadcast new mappings and "steal" IP address
 - ▶ Some implementations do not accept gratuitous ARP messages

Attacks to RARP

- ▶ RARP, as ARP, does not provide any authentication mechanisms
- ▶ An attacker can race against legitimate servers sending fake replies
- ▶ By doing this, an attacker can assign the IP address of an existing host to a particular diskless workstation cutting out the victim host from traffic

Tools

- ▶ `arp`
 - ▶ Used to manipulate the system ARP cache
- ▶ `ifconfig`
 - ▶ Used to configure/monitor a network interface
- ▶ `arpwatch`
 - ▶ Arpwatch uses libpcap to listen for ARP packets on a local Ethernet interface
 - ▶ Arpwatch keeps track for Ethernet/IP address pairs
 - ▶ It syslogs activity and reports certain changes via email

Wait a Minute! What About Switched Ethernet?

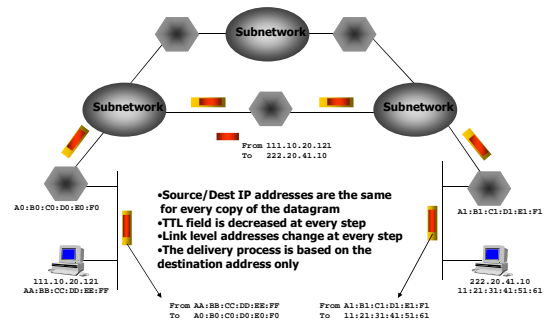
- ▶ Switched Ethernet does not allow direct sniffing
- ▶ ARP spoofing with forwarding can be used to bypass this protection
- ▶ MAC flooding
 - ▶ Switches maintain a table with MAC address/port mappings
 - ▶ Flooding the switch with bogus MAC addresses will overflow table memory and revert the behavior from "switch" to "hub"
- ▶ MAC duplicating/cloning
 - ▶ You reconfigure your host to have the same MAC address as the machine whose traffic you're trying to sniff
 - ▶ The switch will record this in its table and send the traffic to you

Attacks at Network Layer and Above

Routing: Indirect Delivery

- ▶ If two hosts are in different physical networks, the IP datagram is encapsulated in a lower level protocol and delivered to the directly connected gateway
- ▶ The gateway decides which is the next step in the delivery process
- ▶ This step is repeated until a gateway that is in the same physical subnetwork of the destination host is reached
- ▶ Then direct delivery is used

Routing



Types of Routing

- ▶ Source routing
 - ▶ The originator of a datagram determines the route to follow independently before sending the datagram (IP source routing option)
- ▶ Hop-by-hop routing
 - ▶ The delivery route is determined by the gateways that participate in the delivery process

Attacks Using Source Routing

- ▶ The IP source routing option can be used to specify the route to be used in the delivery process, independent of the “normal” delivery mechanisms
- ▶ Using source routing a host can force the traffic through specific routes that allow one to access the traffic (to perform sniffing or man-in-the-middle attacks)
- ▶ If the reverse route is used to reply to traffic, a host can easily impersonate another host that has some kind of privileged relationship with the host that is the destination of the datagram (a trust relationship)

Hop-by-hop Routing: The Routing Table

- ▶ The information about delivery is maintained in the *routing table*

```
% route -n
Kernel IP Routing table
Destination Gateway Genmask Flags Iface
192.168.1.24 0.0.0.0 255.255.255.255 UH eth0
192.168.1.0 0.0.0.0 255.255.255.0 U eth0
127.0.0.0 0.0.0.0 255.0.0.0 U lo
0.0.0.0 192.168.1.1 0.0.0.0 UG eth0
```

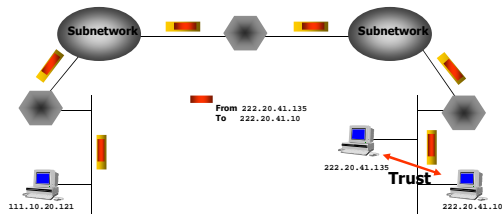
- ▶ Flags
 - ▶ U: the route is up
 - ▶ G: the destination is a gateway
 - ▶ H: the route is to a host (if not set, the route is to a network)
 - ▶ D: the route was created by a *redirect* message
 - ▶ M: the route was modified by a *redirect* message

Routing Mechanism

- ▶ Search for a matching host address
- ▶ Search for a matching network address
- ▶ Search for a default entry
- ▶ If a match is not found a message of “host unreachable” or “network unreachable” is returned (by the kernel or by a remote gateway by using ICMP)
- ▶ Routing tables can be set
 - ▶ Statically (at startup, or by using the “route” command)
 - ▶ Dynamically (using *routing protocols*)

Blind IP Spoofing

- ▶ A host sends an IP datagram with the address of some other host as the source address
- ▶ The host replies to the legitimate host
- ▶ Usually the attacker does not have access to the reply traffic



Man-in-the-middle Attacks

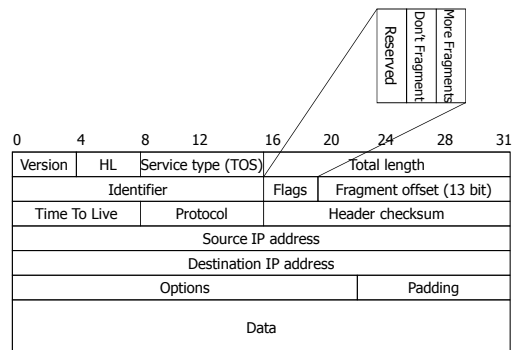
- ▶ An attacker that has control a gateway used in the delivery process can
 - ▶ Sniff the traffic
 - ▶ Intercept/block traffic
 - ▶ Modify traffic



Fragmentation

- ▶ When a datagram is encapsulated in lower level protocols (e.g., Ethernet) it may be necessary to split the datagram in smaller portions
- ▶ This happens when the datagram size is bigger than the data link layer MTU (Maximum Transmission Unit)
- ▶ Fragmentation can be performed at the source host or at an intermediate step in datagram delivery
- ▶ If the datagram has the “do not fragment” flag set, an ICMP error message is sent back to the originator

IP Datagram



Fragmentation

- ▶ If the datagram can be fragmented:
 - ▶ The header is copied in each fragment
 - ▶ In particular, the “datagram id” is copied in each fragment
 - ▶ If the “more fragments” flag is set with the exception of the last fragment
 - ▶ The “fragmentation offset” field contains the position of the fragment with respect to the original datagram expressed in 8 byte units
 - ▶ The “total length field” is changed to match the size of the fragment
- ▶ Each fragment is then delivered as a separate datagram
- ▶ If one fragment is lost the entire datagram is discarded after a timeout

Fragmentation

```

09:52:32.150083 < 192.168.48.69 > 192.168.48.70: (frag 36542:920831080)
09:52:32.151231 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480826600+)
09:52:32.152483 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480829120+)
09:52:32.153703 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480826640+)
09:52:32.154896 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480825160+)
09:52:32.156208 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480823680+)
09:52:32.157401 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480822200+)
09:52:32.158632 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480820720+)
09:52:32.160441 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480817760+)
09:52:32.161640 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480816280+)
09:52:32.162951 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480814800+)
09:52:32.164133 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480813320+)
09:52:32.165379 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480811840+)
09:52:32.166559 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480810360+)
09:52:32.167797 < 192.168.48.69 > 192.168.48.70: (frag 36542:148088880+)
09:52:32.169107 < 192.168.48.69 > 192.168.48.70: (frag 36542:148087400+)
09:52:32.170884 < 192.168.48.69 > 192.168.48.70: (frag 36542:148084440+)
09:52:32.172114 < 192.168.48.69 > 192.168.48.70: (frag 36542:148082960+)
09:52:32.173296 < 192.168.48.69 > 192.168.48.70: (frag 36542:1480814800+)
09:52:32.174527 < 192.168.48.69 > 192.168.48.70: icmp: echo request (frag 36542:1480800+)
    
```

Fragmentation Attacks: Ping of Death

- ▶ The offset of the last segment is such that the total size of the reassembled datagram is bigger than the maximum allowed size
- ▶ A kernel static buffer is overflowed, causing a kernel panic

Ping of Death

```

23:01:06.266645 < 192.168.48.69 > 192.168.48.70: icmp: echo request (frag 4321:148080+)
23:01:06.421261 < 192.168.48.69 > 192.168.48.70: (frag 4321:148082960+)
23:01:06.575953 < 192.168.48.69 > 192.168.48.70: (frag 4321:148084960+)
23:01:06.730065 < 192.168.48.69 > 192.168.48.70: (frag 4321:148086960+)
23:01:06.884625 < 192.168.48.69 > 192.168.48.70: (frag 4321:148088920+)
23:01:07.038801 < 192.168.48.69 > 192.168.48.70: (frag 4321:148090880+)
23:01:07.193403 < 192.168.48.69 > 192.168.48.70: (frag 4321:148092880+)
23:01:07.348185 < 192.168.48.69 > 192.168.48.70: (frag 4321:148094840+)
23:01:07.502326 < 192.168.48.69 > 192.168.48.70: (frag 4321:148096800+)
[...]
23:01:12.451121 < 192.168.48.69 > 192.168.48.70: (frag 4321:148098720+)
23:01:12.605235 < 192.168.48.69 > 192.168.48.70: (frag 4321:148099680+)
23:01:12.759927 < 192.168.48.69 > 192.168.48.70: (frag 4321:148100640+)
23:01:12.917811 < 192.168.48.69 > 192.168.48.70: (frag 4321:148101600+)
23:01:13.090936 < 192.168.48.69 > 192.168.48.70: (frag 4321:148102560+)

```

Total 65120 + 398 = 65518 + 20 bytes of header = 65538 > 65535!

Fragmentation Attacks: Teardrop

- ▶ Denial-of-service attack that exploits a bug in the fragment reassembling routines

count is the number of bytes copied so far
 skb->len is the total size of the datagram (after reassembly)
 ptr is the memory buffer where the datagram is being reassembled
 qp->fragments is the linked list of fragments

```

fp = qp->fragments;
while(fp != NULL) {
    if(count + fp->len > skb->len)
        {error_too_big;} /* We don't want a ping of death :) */
    memcpy(ptr + fp->offset, fp->ptr, fp->len);
    count += fp->len;
    fp = fp->next;
}

```

- ▶ How is `fp->len` computed?

Fragmentation Attacks: Teardrop

- ▶ Computes the positioning of the fragment
`end = fp->offset + ntohs(fp->iph->tot_len) - fp->ihl;`
- ▶ If there are overlapping fragments, realigns
`if (fp->prev != NULL && fp->offset < fp->prev->end) {`
`overlap = fp->prev->end - fp->offset; /* size of the overlap */`
`offset += overlap; /* increment ptr into datagram */`
`fp->ptr += overlap; /* increment ptr into fragment data */ }`
- ▶ Initializes the fragment data structure
`fp->offset = offset;`
`fp->end = end;`
`fp->len = end - offset;`
- ▶ If the fragment is smaller than the size of the overlap `offset` will be bigger than `end` and `fp->len` will be negative!

Fragmentation Attacks: Teardrop

```

222.222.222.222 > 192.168.48.50: (frag 242:4080+)
222.222.222.222 > 192.168.48.50: (frag 242:8024)

offset = 24
end = 24 + 28 - 20 = 32
prev->end = 40 > offset
overlap = 40 - 24 = 16
offset = offset + overlap = 40
fp->len = end - offset = 32 - 40 = -8

```

- ▶ `fp->len` will be interpreted as an unsigned value (65528)
- ▶ Too many bytes will be copied and the kernel will crash

Fragmentation Attacks: Evasion and Denial-Of-Service

- ▶ Firewalls and intrusion detection systems analyze incoming datagrams using the information contained in both the datagram header and the datagram payload (TCP ports, UDP ports, SYN and ACK flags in the TCP header)
- ▶ An attacker may use fragmentation to avoid filtering
 - ▶ Some firewalls make a decision on the first fragment and let the other fragments through by keeping track of the datagram ID
 - ▶ The first fragment of a TCP-over-IP may contain only 8 bytes (source and destination ports for both UDP and TCP)
 - ▶ Setup flags (SYN/ACK) can be "postponed" so that incoming SYN can go through
 - ▶ Setup flags can be overwritten by using overlapping fragments
 - ▶ In some cases, even the original src/dst port can be rewritten

Fragmentation Attacks: Evasion and Denial-Of-Service

- ▶ An attacker may use fragmentation to avoid detection
 - ▶ Some network-based IDSs do not reassemble datagrams
 - ▶ An IDS may obtain a reassembled datagram different from the one obtained at the receiving host
- ▶ An attacker may use fragmentation to build a DOS attack
 - ▶ An attacker may exploit problems in the reassembling code (teardrop, ping of death)
 - ▶ If firewalls and IDSs reassemble a datagram before analyzing it, it is possible to force the system to use a large amount of memory
- ▶ **Fragrouter** is a tool that allows one to send traffic fragmented in different (weird) ways

Broadcasting and Multicasting

- ▶ **Limited broadcast:** all the hosts in the local subnet
 - ▶ IP destination address: 255.255.255.255
 - ▶ Ethernet destination address: ff:ff:ff:ff:ff:ff
 - ▶ These packets are never forwarded (that is they don't "spill" outside the physical link)
- ▶ **Net-directed broadcasts:** all the hosts in a remote subnet
 - ▶ Host ID of all 1s (need to check the netmask)
 - ▶ IP destination address: XX.XX.XX.255
 - ▶ Example valid if mask is 255.255.255.0
- ▶ **Multicast communication:** a subset of the host in a network

Multicast Addresses

- ▶ **Multicast IP addresses:** 1110-xxx...xxx
 - ▶ Range 224.0.0.0 – 239.255.255.255
 - ▶ xxx...xxx (28 bits) represent the multicast group ID
- ▶ Hosts can "register" to receive multicast traffic for a specific group ID
- ▶ Some group ID are pre-defined
 - ▶ 224.0.0.1 (all hosts on the local subnet)
 - ▶ 224.0.0.2 (all routers on the local subnet)
- ▶ **Ethernet multicast addresses:** 00:00:5e:xx:xx:xx:xx
 - ▶ Range 00:00:5e:00:00:00:00 - 00:00:5e:7f:ff:ff:ff
- ▶ The last 23 bits in the IP/Ethernet multicast addresses are the same

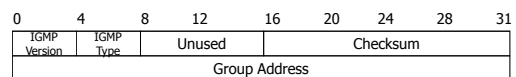
Direct Delivery of Multicast Traffic

- ▶ IP datagrams with a destination multicast address are encapsulated into Ethernet frames with the corresponding Ethernet multicast address
- ▶ The receiver(s) that registered for the particular group ID will receive the datagram

Indirect Delivery of Multicast Traffic

- ▶ A router has to know if there are hosts in the local subnet interested in multicast traffic
- ▶ Hosts "join" and "leave" multicast groups using IGMP (Internet Group Management Protocol)
- ▶ Routers create a delivery path through the Internet using routing protocols such as
 - ▶ PIM (Protocol Independent Multicast)
 - ▶ Helps build multicast forwarding trees
 - ▶ Routers maintain multicast states to remember which forwarding trees they are part of
 - ▶ MSDP (Multicast Source Distribution Protocol)
 - ▶ ...

IGMP



- ▶ **Type**
 - ▶ 1: query sent by a router (on a regular basis)
 - ▶ Destination of IP datagram: 224.0.0.1
 - ▶ Group address in IGMP packet set to 0
 - ▶ 2: report sent by a host (when joining or as a reaction to a query)
 - ▶ Destination of IP datagram: the desired multicast group ID
 - ▶ Group address in IGMP set to the desired multicast group ID
- ▶ A host leaves a group by stopping to answer to router queries

Multicast Attacks

- ▶ By advertising many sources of multicast traffic one site may cause the generation of a “flood” of multicast routing messages
- ▶ The Ramen worm had an erroneous module for IP scans that would use multicast addresses as normal addresses
- ▶ Each scan packet caused the generation of a Multicast Source Discovery Protocol (MSDP) Source Availability (SA) message
- ▶ Ramen generated approx. 65000 SA messages in about 15 minutes
- ▶ The SA messages were flooded throughout the multicast backbone...

Multicast Attacks

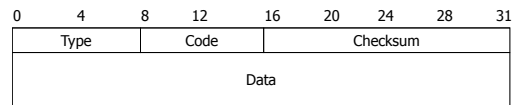
- ▶ PIM Join mechanism can be abused to launch a multicast state saturation attack
 - ▶ A host H can join to many multicast groups for a specific source S
 - ▶ Each group causes routers on H-to-S path to create multicast state to keep track of multicast trees that they are part of
 - ▶ Multicast state buffer at routers are of finite size
 - ▶ At one point, the buffer fills up and the router cannot support any other (legitimate) multicast join requests
- ▶ Discuss Kurian & Sarac ICCCN'06 paper on this

80

Internet Control Message Protocol

- ▶ ICMP is used to exchange control/error messages about the delivery of IP datagrams
- ▶ ICMP messages are encapsulated inside IP datagrams
- ▶ ICMP messages can be:
 - ▶ Requests
 - ▶ Responses
 - ▶ Error messages
 - ▶ An ICMP error message includes the header and a portion of the payload (usually the first 8 bytes) of the offending IP datagram

Message Format



ICMP Messages

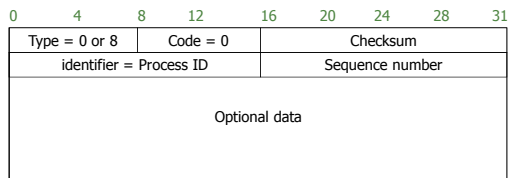
- ▶ *Address mask request/reply*: used by diskless systems to obtain the network mask at boot time
- ▶ *Timestamp request/reply*: used to synchronize clocks
- ▶ *Source quench*: used to inform about traffic overloads
- ▶ *Parameter problem*: used to inform about errors in the IP datagram fields

ICMP Messages

- ▶ *Echo request/reply*: used to test connectivity (*ping*)
- ▶ *Time exceeded*: used to report expired datagrams (TTL = 0)
- ▶ *Redirect*: used to inform hosts about better routes (gateways)
- ▶ *Destination unreachable*: used to inform a host of the impossibility to deliver traffic to a specific destination

ICMP Echo Request/Reply

- Used by the *ping* program



Ping

```
# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) from 192.168.1.100 : 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=1.049 msec
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=660 usec
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=597 usec
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=548 usec
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=601 usec
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=592 usec
64 bytes from 192.168.1.1: icmp_seq=6 ttl=64 time=547 usec

--- 192.168.1.1 ping statistics ---
7 packets transmitted, 7 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.547/0.656/1.049/0.165 ms
```

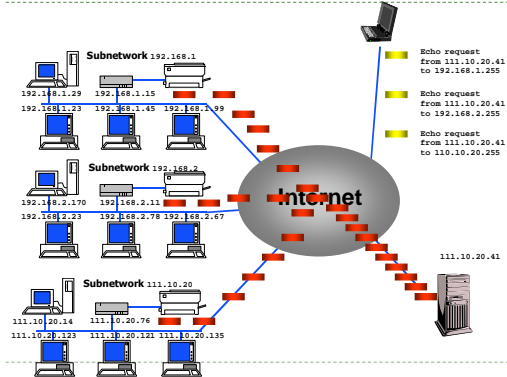
ICMP Echo Attacks

- ICMP Echo Request messages can be used to map the hosts of a network (pingscan or ipsweep)
- ICMP echo datagrams are sent to all the hosts in a subnetwork
- The attacker collects the replies and determines which hosts are actually alive

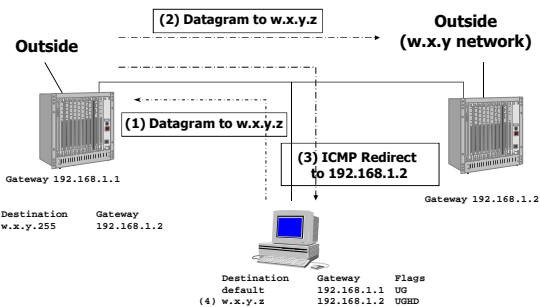
```
Starting nmap by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
Host cisco-sales.ns.com (192.168.31.11) appears to be up.
Host sales1.ns.com (192.168.31.19) appears to be up.
Host sales4.ns.com (192.168.31.22) appears to be up.
Host sales2.ns.com (192.168.31.43) appears to be up.
Host sales3.ns.com (192.168.31.181) appears to be up.
```

- Nmap run completed -- 256 IP addresses (5 hosts up) scanned in 1 second
- ICMP Echo Request can be used to perform a denial of service attack (smurf)

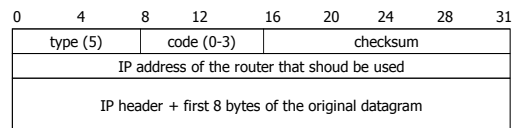
Smurf



ICMP Redirect



ICMP Redirect



- Code
 - 0: redirect for networks (deprecated and usually treated as host)
 - 1: redirect for hosts

ICMP Redirect

- ▶ A host that receives an ICMP redirect message performs the following checks:
 - ▶ The new router must be on a directly connected network
 - ▶ The redirect must be from the current router for that destination
 - ▶ The redirect cannot tell the host to use itself as the router
 - ▶ The route that is being modified must be an indirect route

ICMP Redirect Attacks

- ▶ ICMP *redirect* messages can be used to re-route traffic on specific routes or to a specific host that is not a router at all
- ▶ The attack is performed sending to a host a spoofed ICMP redirect message that appears to come from the host's default gateway
- ▶ The attack can be used to
 - ▶ Hijack traffic
 - ▶ Perform a denial-of-service attack

ICMP Redirect Attacks

```
# arp -n
Address          HWtype  HWaddress
192.168.1.1      ether   00:20:78:CA:7E:A8
192.168.1.10     ether   00:01:03:1D:98:B8
192.168.1.100    ether   08:00:46:07:04:A3

C:\WINDOWS>route PRINT
Active Routes:
Network Address      Netmask  Gateway Address  Interface  Metric
0.0.0.0              0.0.0.0  192.168.1.1      192.168.1.10  1
127.0.0.0            255.0.0.0  127.0.0.1        127.0.0.1    1
192.168.1.0          255.255.255.0  192.168.1.10    192.168.1.10  1
192.168.1.10        255.255.255.255  127.0.0.1        127.0.0.1    1
192.168.1.255       255.255.255.255  192.168.1.10    192.168.1.10  1

# tcpdump -n
8:0:46:7:4:a3 0:1:3:1d:98:b8 0800 70: 192.168.1.1 > 192.168.1.10:
icmp: redirect 128.111.48.69 to host 192.168.1.10
```

ICMP Redirect Attacks

```
C:\WINDOWS>route PRINT
Active Routes:
Network Address      Netmask  Gateway Address  Interface  Metric
0.0.0.0              0.0.0.0  192.168.1.1      192.168.1.10  1
127.0.0.0            255.0.0.0  127.0.0.1        127.0.0.1    1
128.111.48.69       255.255.255.255  192.168.1.100    192.168.1.10  1
192.168.1.0          255.255.255.0  192.168.1.10    192.168.1.10  1
192.168.1.10        255.255.255.255  127.0.0.1        127.0.0.1    1
192.168.1.255       255.255.255.255  192.168.1.10    192.168.1.10  1

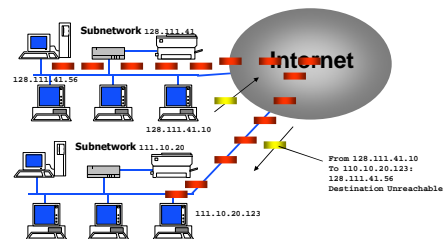
C:\WINDOWS>ping 128.111.48.69
0:1:3:1d:98:b8 8:0:46:7:4:a3 0800 74: 192.168.1.10 > 128.111.48.69:
icmp: echo request
0:1:3:1d:98:b8 8:0:46:7:4:a3 0800 74: 192.168.1.10 > 128.111.48.69:
icmp: echo request
...
```

ICMP Destination Unreachable

- ▶ ICMP message used by gateways to state that the datagram cannot be delivered
- ▶ Many subtypes
 - ▶ Network unreachable
 - ▶ Host unreachable
 - ▶ Protocol unreachable
 - ▶ Port unreachable
 - ▶ Fragmentation needed but don't fragment bit set
 - ▶ Destination host unknown
 - ▶ Destination network unknown
 - ▶ ...

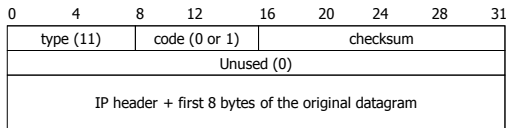
Destination Unreachable Attacks

- ▶ *Forged destination unreachable* messages can cut out nodes from the network (denial of service)



ICMP Time Exceeded

- Used when
 - TTL becomes zero (code = 0)
 - The reassembling of a fragmented datagram times out (code =1)



Traceroute

- ICMP Time Exceeded messages are used by the traceroute program to determine the path used to deliver a datagram
- A series of IP datagrams are sent to the destination node
- Each datagram has an increasing TTL field (starting at 1)
- From the ICMP *Time exceeded* messages returned by the intermediate gateways it is possible to reconstruct the route from the source to the destination
- Note: traceroute allows one to specify loose source routing (-g option)
- Useful for network analysis, topology mapping

Traceroute

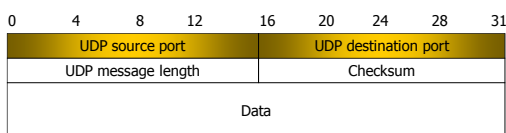
```

C:\Documents and Settings\kxs028100>tracert -d www.google.com
Tracing route to www.l.google.com [74.125.45.147] over a maximum of 30 hops:
 0  0 ms  0 ms  0 ms  10.25.20.50
 1  2 ms  2 ms  2 ms  10.21.0.1
 2  2 ms  3 ms  2 ms  129.110.5.87
 3  3 ms  2 ms  3 ms  129.110.5.65
 4  3 ms  3 ms  3 ms  208.76.224.73
 5  3 ms  3 ms  4 ms  38.104.34.25
 6  4 ms  4 ms  3 ms  154.54.0.125
 7  4 ms  4 ms  5 ms  154.54.5.126
 8  14 ms 14 ms 14 ms 154.54.2.234
 9  31 ms 103 ms 32 ms 154.54.6.206
10  27 ms 26 ms 27 ms 154.54.12.130
11  25 ms 25 ms 25 ms 209.85.254.128
12  27 ms 26 ms 25 ms 72.14.236.26
13  35 ms 34 ms 36 ms 72.14.232.213
14  35 ms 69 ms 34 ms 209.85.253.137
15  48 ms 42 ms 36 ms 74.125.45.147
16  35 ms 36 ms 36 ms
Trace complete.
C:\Documents and Settings\kxs028100>
    
```

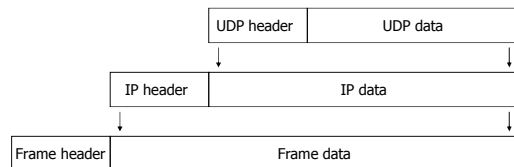
User Datagram Protocol (UDP)

- The UDP protocol relies on IP to provide a *connectionless, unreliable, best-effort* datagram delivery service (delivery, integrity, non-duplication, ordering, and bandwidth is not guaranteed)
- Introduces the *port* abstraction that allows one to address different message destinations for the same IP address
- Often used for multimedia (more efficient than TCP) and for services based on request/reply schema (DNS, NIS, NFS, RPC)

UDP Message

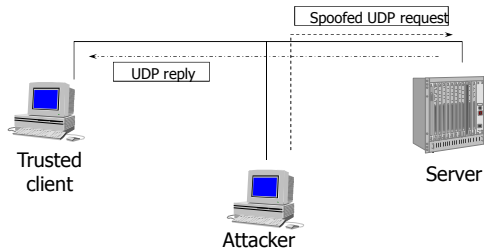


UDP Encapsulation



UDP Spoofing

- ▶ Basically IP spoofing
- ▶ Very easy to perform



UDP Spoofing

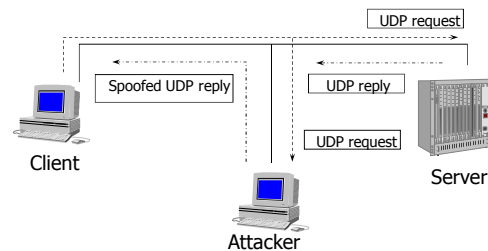
- ▶ Protocols based on UDP that are vulnerable to spoofing attacks
 - ▶ DNS
 - ▶ RPC
 - ▶ NFS
 - ▶ NIS
 - ▶ ...

Using Spoofing to Create UDP Storms

- ▶ A spoofed UDP datagram is sent to the echo service (7)
- ▶ The source port is set to the chargen service (19)
- ▶ The reply of the echo service is interpreted as a request by the chargen service
- ▶ The reply of the chargen service is interpreted as a request by the echo service
- ▶ ...
- ▶ The same attack can be carried out using two echo services

UDP Hijacking

- ▶ Variation of the UDP spoofing attack



UDP Portscan

- ▶ Used to determine which UDP services are available
- ▶ A zero-length UDP packet is sent to each port
- ▶ If an ICMP error message "port unreachable" is received the service is assumed to be unavailable
- ▶ Many TCP/IP stack implementations (not Windows!) implement a limit on the error message rate, therefore this type of scan can be *slow* (e.g., Linux limit is 80 messages every 4 seconds)

UDP Portscan

```
% nmap -sU 192.168.1.10

Starting nmap by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on (192.168.1.10):
(The 1445 ports scanned but not shown below are in state: closed)
Port      State  Service
137/udp   open   netbios-ns
138/udp   open   netbios-dgm

Nmap run completed -- 1 IP address (1 host up) scanned in 4 seconds
```

UDP Portscan

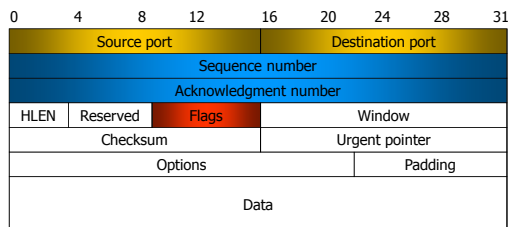
```

19:37:31.305674 192.168.1.100.41481 > 192.168.1.10.134: udp 0 (ttl 46, id 61284)
19:37:31.305706 192.168.1.100.41481 > 192.168.1.10.134: udp 0 (ttl 46, id 31166)
19:37:31.305730 192.168.1.100.41481 > 192.168.1.10.137: udp 0 (ttl 46, id 31406)
19:37:31.305734 192.168.1.100.41481 > 192.168.1.10.140: udp 0 (ttl 46, id 50734)
19:37:31.305770 192.168.1.100.41481 > 192.168.1.10.131: udp 0 (ttl 46, id 33361)
19:37:31.305775 192.168.1.100.41481 > 192.168.1.10.132: udp 0 (ttl 46, id 14242)
19:37:31.305804 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 134 unreachable
19:37:31.305809 192.168.1.100.41481 > 192.168.1.10.135: udp 0 (ttl 46, id 17622)
19:37:31.305815 192.168.1.100.41481 > 192.168.1.10.139: udp 0 (ttl 46, id 52452)
19:37:31.305871 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 140 unreachable
19:37:31.305875 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 131 unreachable
19:37:31.305881 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 132 unreachable
19:37:31.305887 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 135 unreachable
19:37:31.305892 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 139 unreachable
19:37:31.305927 192.168.1.100.41481 > 192.168.1.10.133: udp 0 (ttl 46, id 38693)
19:37:31.305932 192.168.1.100.41481 > 192.168.1.10.130: udp 0 (ttl 46, id 60943)
19:37:31.305974 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 133 unreachable
19:37:31.305979 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 130 unreachable
19:37:31.617611 192.168.1.100.41482 > 192.168.1.10.138: udp 0 (ttl 46, id 21936)
19:37:31.617641 192.168.1.100.41482 > 192.168.1.10.137: udp 0 (ttl 46, id 17647)
19:37:31.617663 192.168.1.100.41481 > 192.168.1.10.136: udp 0 (ttl 46, id 55)
19:37:31.617797 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 136 unreachable
    
```

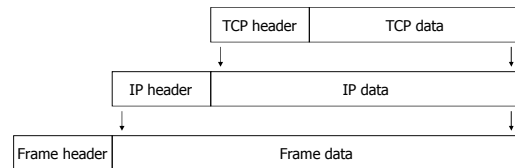
Transmission Control Protocol (TCP)

- ▶ The TCP protocol relies on IP to provide a *connection-oriented, reliable* stream delivery service (no loss, no duplication, no transmission errors, correct ordering)
- ▶ TCP, as UDP, provides the *port* abstraction
- ▶ TCP allows two nodes to establish a *virtual circuit*, identified by source IP address, destination IP address, source TCP port, destination TCP port
- ▶ The virtual circuit is composed of two *streams* (full-duplex connection)
- ▶ The couple IP address/port number is sometimes called a *socket* (and the two streams are called a *socket pair*)

TCP Segment



TCP Encapsulation



TCP Seq/Ack Numbers

- ▶ The *sequence number* specifies the position of the segment data in the communication *stream* (SYN=13423 means: *the payload of this segment contains the data from byte 13423 to byte 13458*)
- ▶ The acknowledgment number specifies the position of the next byte expected from the communication partner (ACK = 16754 means: *I have received correctly up to byte 16753 in the stream, I expect the next byte to be 16754*)
- ▶ These numbers are used to manage retransmission of lost segments, duplication, flow control

TCP Window

- ▶ The TCP window is used to perform flow control
- ▶ Segment will be accepted only if their sequence numbers are inside the window that starts with the current acknowledgment number: $\text{ack number} < \text{sequence number} < \text{ack number} + \text{window}$
- ▶ The window size can change dynamically to adjust the amount of information sent by the sender

TCP Flags

- ▶ Flags are used to manage the establishment and shutdown of a virtual circuit
 - ▶ SYN: request for the synchronization of syn/ack numbers (used in connection setup)
 - ▶ ACK: states the acknowledgment number is valid (all segment in a virtual circuit have this flag set, except for the first one)
 - ▶ FIN: request to shutdown one stream
 - ▶ RST: request to immediately reset the virtual circuit
 - ▶ URG: states that the Urgent Pointer is valid
 - ▶ PSH: request a “push” operation on the stream (that is, the stream data should be passed to the user application as soon as possible)

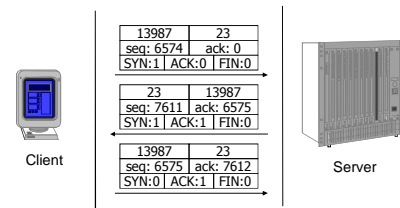
TCP Virtual Circuit: Setup

- ▶ A *server*, listening to a specific *port*, receives a connection request from a *client*: The segment containing the request is marked with the *SYN* flag and contains a random initial sequence number s_c
- ▶ The server answers with a segment marked with both the *SYN* and *ACK* flags and containing
 - ▶ an initial random sequence number s_s
 - ▶ $s_c + 1$ as the acknowledgment number
- ▶ The client sends a segment with the *ACK* flag set and with sequence number $s_c + 1$ and acknowledgment number $s_s + 1$

What Initial Sequence Number?

- ▶ The TCP standard (RFC 793) specifies that the sequence number should be incremented every 4 microseconds
- ▶ BSD UNIX systems initially used a number that is incremented by 64,000 every half second (8 microseconds increments) and by 64,000 each time a connection is established

TCP: Three-way Handshake



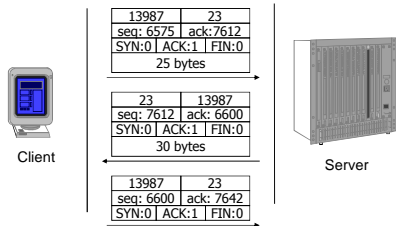
TCP: Three-way Handshake

```
arp who-has 192.168.1.20 tell 192.168.1.10
arp reply 192.168.1.20 is-at 0:10:4b:e2:f6:4c
192.168.1.10.1026 > 192.168.1.20.23: S 1015043:1015043(0)
192.168.1.20.23 > 192.168.1.10.1026: S 4056577943:4056577943(0) ack 1015044
192.168.1.10.1026 > 192.168.1.20.23: . ack 4056577944
```

TCP Virtual Circuit: Data Exchange

- ▶ A partner sends in each packet the acknowledgment of the previous segment and its own sequence number increased of the number of transmitted bytes
- ▶ A partner accepts a segment of the other partner only if the numbers are inside the transmission window
- ▶ An empty segment may be used to acknowledge the received data

TCP Virtual Circuit: Data Exchange



TCP Virtual Circuit: Data Exchange

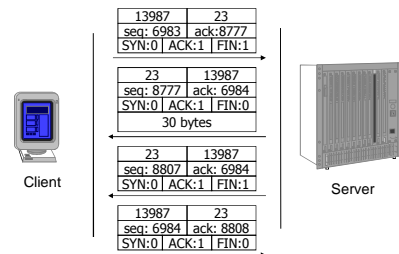
```

192.168.1.20.23 > 192.168.1.10.1026: P 4056577944:4056577956(12) ack 1015044
192.168.1.10.1026 > 192.168.1.20.23: P 1015044:1015047(3) ack 4056577956
192.168.1.20.23 > 192.168.1.10.1026: . ack 1015047
192.168.1.10.1026 > 192.168.1.20.23: P 1015047:1015056(9) ack 4056577956
192.168.1.20.23 > 192.168.1.10.1026: P 4056577956:4056577962(6) ack 1015056
192.168.1.10.1026 > 192.168.1.20.23: P 1015056:1015066(10) ack 4056577962
192.168.1.20.23 > 192.168.1.10.1026: . ack 1015066
192.168.1.20.23 > 192.168.1.10.1026: P 4056577962:4056577977(15) ack 1015066
192.168.1.10.1026 > 192.168.1.20.23: P 1015066:1015069(3) ack 4056577977
192.168.1.20.23 > 192.168.1.10.1026: . ack 1015069
    
```

TCP Virtual Circuit: Shutdown

- ▶ One of the partners, say A, can terminate its stream by sending a segment with the FIN flag set
- ▶ The other partner, say B, answers with an ACK segment
- ▶ From that point on, A will not send any data to B: it will just acknowledge data sent by B
- ▶ When B shutdowns its stream the virtual circuit is considered closed

TCP Virtual Circuit: Shutdown



TCP Virtual Circuit: Shutdown

```

192.168.1.20.23 > 192.168.1.10.1026: F 4056579200:4056579200(0) ack 1016070
192.168.1.10.1026 > 192.168.1.20.23: . ack 4056579201
192.168.1.10.1026 > 192.168.1.20.23: F 1016070:1016070(0) ack 4056579201
192.168.1.20.23 > 192.168.1.10.1026: . ack 1016071
    
```

TCP Portscan

- ▶ Used to determine the TCP services available on a victim host
- ▶ Most services are statically associated with port numbers (see /etc/services in UNIX systems)
- ▶ In its simplest form (*connect()* scanning), the attacker tries to open a TCP connection to all the 65535 ports of the victim host
- ▶ If the handshake is successful then the service is available
- ▶ Advantage: no need to be root
- ▶ Disadvantage (?): very noisy

connect() Scan

```
root@localhost/home/user: nmap -sT 192.168.1.20
Starting nmap by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on (192.168.1.20):
(The 1500 ports scanned but not shown below are in state: closed)
Port      State  Service
7/tcp     open   echo
9/tcp     open   discard
11/tcp    open   systat
13/tcp    open   daytime
15/tcp    open   netstat
19/tcp    open   chargen
21/tcp    open   ftp
22/tcp    open   ssh
23/tcp    open   telnet
512/tcp   open   exec
513/tcp   open   login
514/tcp   open   shell
6000/tcp  open   X11

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

TCP SYN Scanning

- ▶ AKA “half-open” scanning
- ▶ The attacker sends a SYN packet
- ▶ If the server answers with a SYN/ACK packet then the port is open or (usually) with a RST packet if the port is closed
- ▶ The attacker sends a RST packet instead of the final ACK
- ▶ The connection is never open and the event is not logged by the operating system/application

TCP SYN Scanning

```
# nmap -sS 192.168.38.78
Port      State  Service
80/tcp    open   http

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second

11:27:32.249220 192.168.48.69.47146 > 192.168.41.38.78: S 3886663922:3886663922(0) win 2048
11:27:32.266910 192.168.48.69.47146 > 192.168.41.38.78: S 3886663922:3886663922(0) win 2048
11:27:32.266914 192.168.48.69.47146 > 192.168.41.38.81: S 3886663922:3886663922(0) win 2048
11:27:32.266918 192.168.48.69.47146 > 192.168.41.38.82: S 3886663922:3886663922(0) win 2048
11:27:32.266923 192.168.48.69.47146 > 192.168.41.38.80: S 3886663922:3886663922(0) win 2048
11:27:32.266925 192.168.48.69.47146 > 192.168.41.38.79: S 3886663922:3886663922(0) win 2048
11:27:32.267904 192.168.41.38.78 > 192.168.48.69.47146: R 0:0(0) ack 3886663923 win 0 (DF)
11:27:32.267970 192.168.41.38.81 > 192.168.48.69.47146: R 0:0(0) ack 3886663923 win 0 (DF)
11:27:32.268038 192.168.41.38.82 > 192.168.48.69.47146: R 0:0(0) ack 3886663923 win 0 (DF)
11:27:32.268106 192.168.41.38.80 > 192.168.48.69.47146: S 1441896698:1441896698(0) ack 3886663923 win 5840 mss 1460> (DF)
11:27:32.268121 192.168.48.69.47146 > 192.168.41.38.80: R 3886663923:3886663923(0) win 0 (DF)
11:27:32.268174 192.168.41.38.79 > 192.168.48.69.47146: R 0:0(0) ack 3886663923 win 0 (DF)
```

TCP FIN Scanning

- ▶ The attacker sends a FIN-marked packet
- ▶ In most TCP/IP implementation (Windows excluded)
 - ▶ If the port is closed a RST packet is sent back
 - ▶ If the port is open the FIN packet is ignored (timeout)
- ▶ Variation of this type of scanning technique
 - ▶ Xmas: FIN, PSH, URG set
 - ▶ Null: no flags set

TCP FIN Scanning

```
# nmap -sF 192.168.41.38

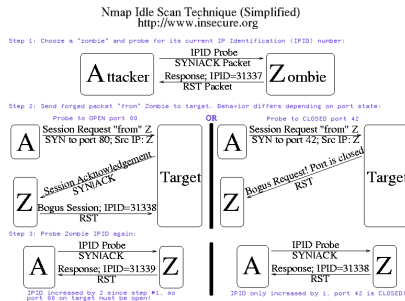
Starting nmap ( www.insecure.org/nmap/ )
Port      State  Service
80/tcp    open   http

11:39:07.356917 192.168.48.69.38772 > 192.168.41.38.79: F 0:0(0) win 1024
11:39:07.356921 192.168.48.69.38772 > 192.168.41.38.82: F 0:0(0) win 1024
11:39:07.356925 192.168.48.69.38772 > 192.168.41.38.81: F 0:0(0) win 1024
11:39:07.356927 192.168.48.69.38772 > 192.168.41.38.80: F 0:0(0) win 1024
11:39:07.356931 192.168.48.69.38772 > 192.168.41.38.78: F 0:0(0) win 1024
11:39:07.357918 192.168.41.38.79 > 192.168.48.69.38772: R 0:0(0) ack 1 win 0 (DF)
11:39:07.357983 192.168.41.38.82 > 192.168.48.69.38772: R 0:0(0) ack 1 win 0 (DF)
11:39:07.358051 192.168.41.38.81 > 192.168.48.69.38772: R 0:0(0) ack 1 win 0 (DF)
11:39:07.358326 192.168.41.38.78 > 192.168.48.69.38772: R 0:0(0) ack 1 win 0 (DF)
11:39:07.666939 192.168.48.69.38773 > 192.168.41.38.80: F 0:0(0) win 1024
11:39:07.976951 192.168.48.69.38772 > 192.168.41.38.80: F 0:0(0) win 1024
11:39:08.286929 192.168.48.69.38773 > 192.168.41.38.80: F 0:0(0) win 1024
```

Idle Scanning

- ▶ Uses a victim host to “relay” the scan
- ▶ The attacker sends spoofed TCP SYN packets to the target
- ▶ The packets appear to come from the victim
- ▶ The target replies to the victim
 - ▶ If the target replies with a SYN+ACK packet (open port) then the victim will send out a reset
 - ▶ If the target replies with a RST (closed port) then the victim will not send out any packet
- ▶ The attacker checks the IP datagram ID of the victim before and after each port probe
 - ▶ If it has increased: port on target was open
 - ▶ If it has not increased: port on target was closed

Idle Scanning with Nmap



OS Fingerprinting

- OS Fingerprinting allows to determine the operating system of a host by examining the reaction to carefully crafted packets
- Wrong answer to FIN TCP packets
- "Undefined" flags in the TCP header of a request are copied verbatim in the reply
- Weird combination of flags in the TCP header
- Selection of TCP initial sequence numbers
- Selection of initial TCP window size
- Analysis of the use of ICMP messages
 - Error rate
 - Amount of offending datagram included
- TCP options

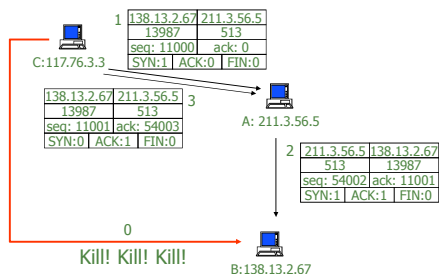
TCP Spoofing

- Attack aimed at impersonating another host when establishing a TCP connection
- First discussed by R.T. Morris in "A Weakness in the 4.2BSD Unix TCP/IP Software" in 1985
- Used by Mitnick in his attack against SDSC

TCP Spoofing

- Node A trusts node B (e.g., login with no password)
- Node C wants to impersonate B with respect to A in opening a TCP connection
- C kills B (flooding, crashing, redirecting) so that B does not send annoying RST segments
- C sends A a TCP SYN segment in a spoofed IP packet with B's address as the source IP and s_s as the sequence number
- A replies with a TCP SYN/ACK segment to B with s_s as the sequence number. B ignores the segment: dead or too busy
- C does not receive this segment but to finish the handshake it has to send an ACK segment with $s_s + 1$ as the acknowledgment number
 - C eavesdrop the SYN/ACK segment
 - C guesses the correct sequence number

TCP Spoofing



TCP Hijacking

- Powerful technique to take control of an existing TCP connection
- The attacker uses spoofed TCP segments to
 - Insert data in the streams
 - Reset an existing connection (denial of service)
- Anyway the correct sequence/acknowledgment numbers must be used
 - The attacker can eavesdrop the traffic between client and server
 - The attacker can guess the correct seq/ack numbers
- Described in "Simple Active Attack Against TCP" by L. Joncheray

TCP Hijacking

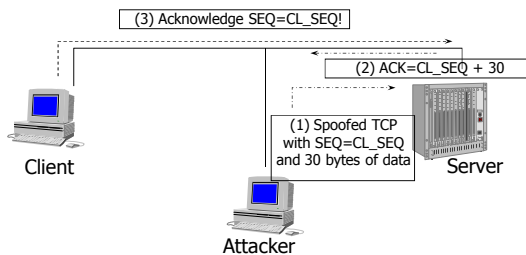
- ▶ The attacker waits until the connection is “quite”
 - ▶ All the transmitted data have been acknowledged (by both endpoints)
- ▶ The attacker injects the data in the stream
 - ▶ “Desynchronize” the connection
- ▶ The receiver of the injected data sends an acknowledgment to the apparent sender
- ▶ The apparent sender replies with an acknowledgement with the “expected” sequence number
- ▶ The receiver considers this as out-of-sync and sends an an acknowledgement with the “expected” sequence number
- ▶ ...

TCP Hijacking

- ▶ ACK messages with no data are not retransmitted in case of loss
- ▶ The “ACK storm” continues until one message is lost
- ▶ Any subsequent attempt to communicate will generate an ACK storm
- ▶ ACK storms can be blocked by the attacker using ACK packets with the right numbers

TCP Hijacking

CL_SEQ = SVR_ACK
SVR_SEQ = CL_ACK



TCP Hijacking

- ▶ This technique can be used against both client and server to completely hijack the communication channel
- ▶ “Early desynchronization” can be achieved by the attacker by resetting existing connections and immediately opening new ones (between the same ports) with different initial sequence numbers

Hunt

- ▶ Hunt is a program that allows one to monitor and hijack TCP connections
- ▶ Available at <http://lin.fsid.cvut.cz/~kra/>

```
# hunt
-- Main Menu -- rcvpkt 26, free/alloc 63/64 -----
l/w/r) list/watch/reset connections
u) host up tests
a) arp/simple hijack (avoids ack storm if arp used)
s) simple hijack
d) daemons rst/arp/sniff/mac
o) options
x) exit
*~
-> w
0) 192.168.1.100 [4169] --> 192.168.1.20 [23]

choose conn> 0
dump [s]rc/[d]st/[b]oth [b]> b
user@lars~/Projects: llss
STAT/ xSTAT/
user@lars~/Projects:
[...]
```

Hunt

```
-- Main Menu -- rcvpkt 0, free/alloc 63/64 -----
l/w/r) list/watch/reset connections
u) host up tests
a) arp/simple hijack (avoids ack storm if arp used)
s) simple hijack
d) daemons rst/arp/sniff/mac
o) options
x) exit
*~ s
0) 192.168.1.10 [1026] --> 192.168.1.20 [23]
choose conn> 0
dump connection y/n [n]>
Enter the command string you wish executed or [cr]> echo "+ +" > .rhosts
ecACK storm detected - reset after 4s
ho "+ +" > .rhosts
hunt: possible ACK storm: 0) 192.168.1.20 [23] --> 192.168.1.10 [1026]
..[user@lars user]$
.....
reset done
```

ACK Storm

```
192.168.1.10.1026 > 192.168.1.20.23: P 1015112:1015133(21) ack 4056578923
192.168.1.20.23 > 192.168.1.10.1026: . seq 4056578923 ack 1015133
192.168.1.10.1026 > 192.168.1.20.23: . seq 1015112 ack 4056578923
192.168.1.20.23 > 192.168.1.10.1026: . seq 4056578923 ack 1015133
192.168.1.10.1026 > 192.168.1.20.23: . seq 1015112 ack 4056578923
192.168.1.20.23 > 192.168.1.10.1026: . seq 4056578923 ack 1015133
192.168.1.10.1026 > 192.168.1.20.23: . seq 1015112 ack 4056578923
192.168.1.20.23 > 192.168.1.10.1026: . seq 4056578923 ack 1015133
192.168.1.10.1026 > 192.168.1.20.23: . seq 1015112 ack 4056578923
192.168.1.20.23 > 192.168.1.10.1026: . seq 4056578923 ack 1015133
192.168.1.10.1026 > 192.168.1.20.23: . seq 1015112 ack 4056578923
192.168.1.20.23 > 192.168.1.10.1026: . seq 4056578923 ack 1015133
192.168.1.10.1026 > 192.168.1.20.23: . seq 1015112 ack 4056578923
192.168.1.20.23 > 192.168.1.10.1026: . seq 4056578923 ack 1015133
192.168.1.10.1026 > 192.168.1.20.23: . seq 1015112 ack 4056578923
192.168.1.20.23 > 192.168.1.10.1026: . seq 4056578923 ack 1015133
192.168.1.10.1026 > 192.168.1.20.23: . seq 1015112 ack 4056578923
192.168.1.20.23 > 192.168.1.10.1026: . seq 4056578923 ack 1015133
```

Syn-flooding Attack

- ▶ Very common denial-of-service attack, aka Neptune
- ▶ Attacker starts handshake with SYN-marked segment
- ▶ Victim replies with SYN-ACK segment
- ▶ Attacker... stays silent
- ▶ A host can keep a limited number of TCP connections in half-open state. After that limit, it cannot accept any more connections
- ▶ Current solutions
 - ▶ Increase the length of the half-open connection queue
 - ▶ Drop half-open connections when the limit has been reached and new requests for connection arrive
 - ▶ Use syn Cookies

Syn Cookies

- ▶ Special algorithm used for determining the initial sequence number of the server
- ▶ The number is
 - ▶ Top 5 bits: $t \bmod 32$, where t is a 32-bit time counter that increases every 64 seconds
 - ▶ Following 3 bits: the encoding of the Maximum Segment Size (MSS) chosen by the server in response to the client's MSS
 - ▶ A keyed hash of:
 - ▶ Counter t
 - ▶ Source/Destination IP addresses and ports

Syn Cookies

- ▶ A server that uses SYN cookies sends back a SYN+ACK, exactly as if the SYN queue had been larger
- ▶ When the server receives an ACK, it checks that the secret function works for a recent value of t , and then rebuilds the SYN queue entry (using the encoded MSS info)
- ▶ Drawbacks:
 - ▶ The server sequence number grows faster than normal
 - ▶ The MSS value is limited by the encoding procedure (only 8 possible values)

Tools

- ▶ Libpcap/libnet: sniffing and injection
- ▶ Libnids: packet reassembling library
- ▶ Dsniff: sniffing, man-in-the-middle attacks
- ▶ Ettercap: sniffer for switched LANs
- ▶ Hunt: hijacking
- ▶ Nmap: scanning tool
 - ▶ IP scans
 - ▶ UDP portscans
 - ▶ TCP portscans
 - ▶ OS fingerprinting
 - ▶ Service fingerprinting

Tools

- ▶ Tcpcat: network sniffer
- ▶ Traceroute: path discovery
- ▶ Hping2: creation of packets from command line
- ▶ Ethereal: a GUI-based sniffer
- ▶ Tcpflow: a TCP flow reassembler
- ▶ Tcpslice: a Tcpcat file slicer
- ▶ Tcptrace: tool for the analysis of Tcpcat files
- ▶ Ngrep: grep-like tool for network packets
- ▶ Ntop: top-like tool for network traffic