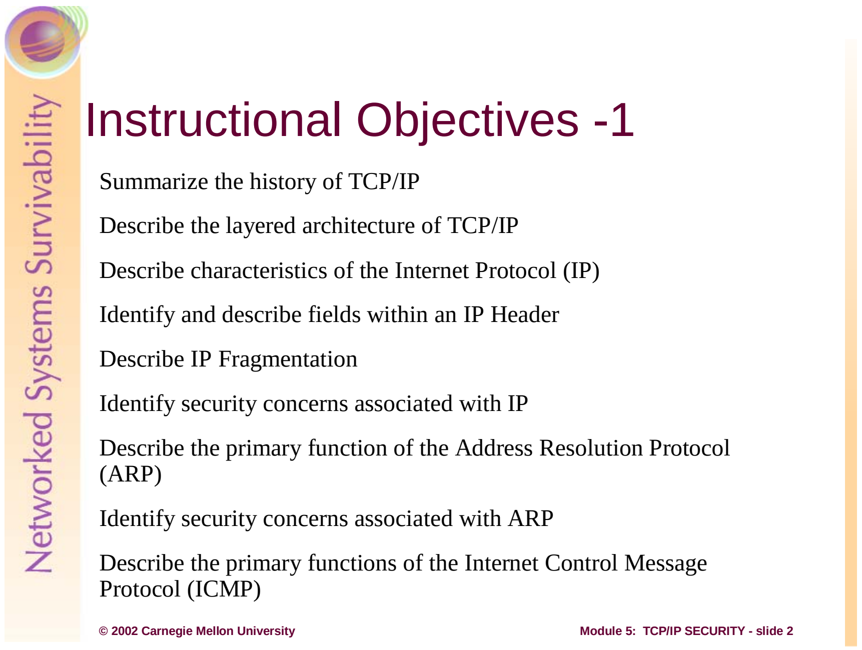
Information Security for Technical Staff

Module 5:

TCP/IP SECURITY

Networked Systems Survivability
CERT® Coordination Center
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890



Instructional Objectives -1

Summarize the history of TCP/IP

Describe the layered architecture of TCP/IP

Describe characteristics of the Internet Protocol (IP)

Identify and describe fields within an IP Header

Describe IP Fragmentation

Identify security concerns associated with IP

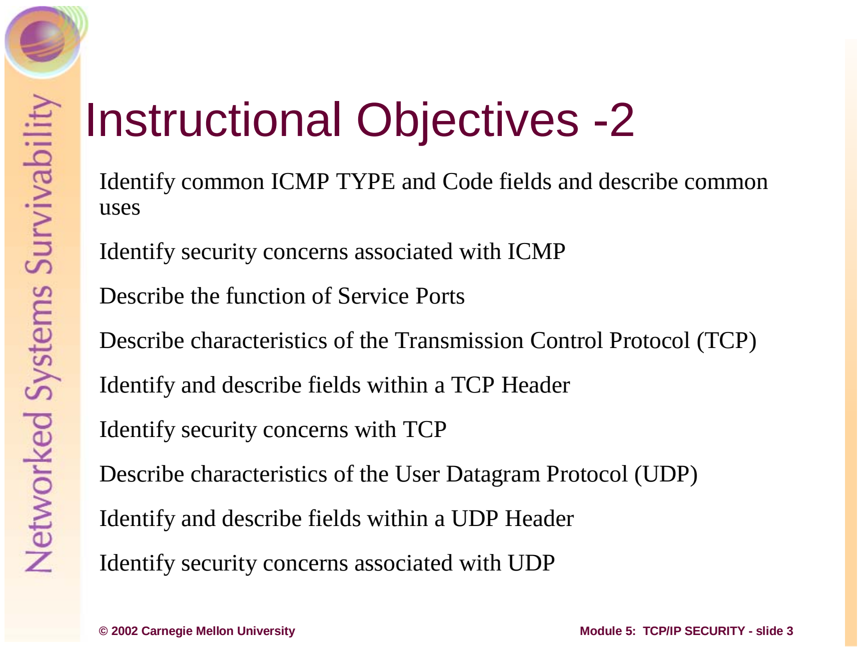
Describe the primary function of the Address Resolution Protocol (ARP)

Identify security concerns associated with ARP

Describe the primary functions of the Internet Control Message Protocol (ICMP)

Networked Systems Survivability
© 2002 Carnegie Mellon University
Module 5: TCP/IP SECURITY - slide 2

Some background knowledge of data networking and the TCP/IP suite is assumed for this portion of the course. Therefore this module does not attempt to provide a complete explanation of the entire protocol suite. Rather, it focuses on the protocols most prevalent in the fields of networked systems survivability and information security.



Networked Systems Survivability

Instructional Objectives -2

Identify common ICMP TYPE and Code fields and describe common uses

Identify security concerns associated with ICMP

Describe the function of Service Ports

Describe characteristics of the Transmission Control Protocol (TCP)

Identify and describe fields within a TCP Header

Identify security concerns with TCP

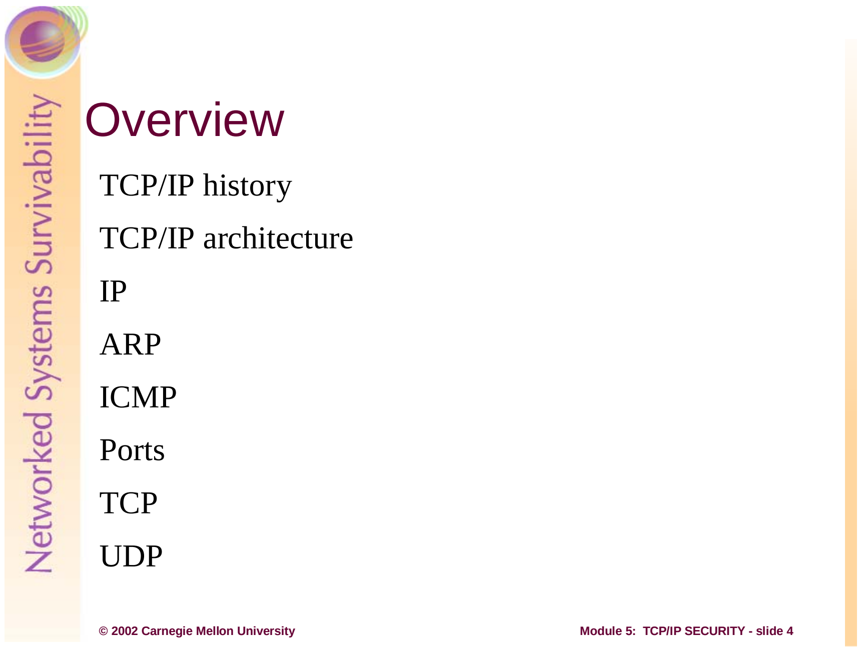
Describe characteristics of the User Datagram Protocol (UDP)

Identify and describe fields within a UDP Header

Identify security concerns associated with UDP

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 3

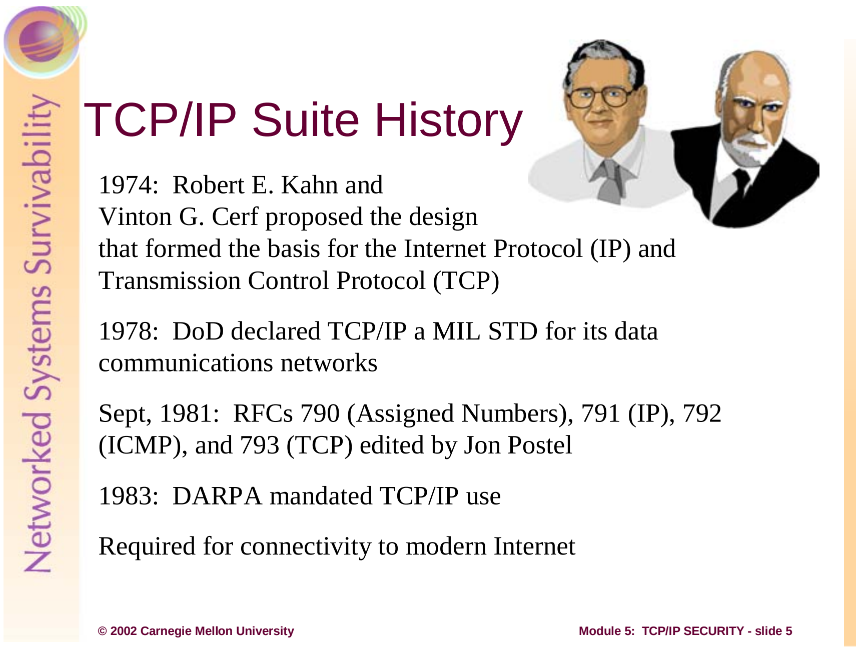


Networked Systems Survivability

Overview

- TCP/IP history
- TCP/IP architecture
- IP
- ARP
- ICMP
- Ports
- TCP
- UDP

© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 4



Networked Systems Survivability

TCP/IP Suite History

1974: Robert E. Kahn and Vinton G. Cerf proposed the design that formed the basis for the Internet Protocol (IP) and Transmission Control Protocol (TCP)

1978: DoD declared TCP/IP a MIL STD for its data communications networks

Sept, 1981: RFCs 790 (Assigned Numbers), 791 (IP), 792 (ICMP), and 793 (TCP) edited by Jon Postel

1983: DARPA mandated TCP/IP use

Required for connectivity to modern Internet

© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 5

The architecture of TCP/IP is often called the Internet architecture because TCP/IP and the Internet are so closely interwoven. Internet standards were developed by the Defense Advanced Research Projects Agency (DARPA) and eventually passed on to the Internet Society.

The Internet was originally proposed as a method of testing the viability of packet-switching networks. During its tenure with the project, DARPA foresaw a network of leased lines connected by switching nodes. The network was called ARPANET, and the switching nodes were called Internet Message Processors, or IMPs. The ARPANET was initially to be comprised of four IMPs located at the University of California at Los Angeles, the University of California at Santa Barbara, the Stanford Research Institute, and the University of Utah. The original IMPs were to be Honeywell 316 minicomputers.

Bolt, Beranek, and Newman (BBN), a company that had a strong influence on the development of the network in the following years, won the contract for the installation of the network. The contract was awarded in late 1968, followed by testing and refinement over the next five years. In 1971, ARPANET entered into regular service. Machines used the ARPANET by connecting to an IMP using the "1822" protocol—so called because that was the number of the technical paper describing the system. During the early years, the purpose and utility of the network was widely discussed, leading to refinements and modifications as users requested more functionality from the system.

A commonly recognized need was the capability to transfer files from one machine to another, as well as the capability to support remote logins. Remote logins would enable a user in Santa Barbara to connect to a machine in Los Angeles over the network and function as though he or she were in front of the UCLA machine. The protocol then in use on the network wasn't capable of handling these new functionality requests, so new protocols were continually developed, refined, and tested.

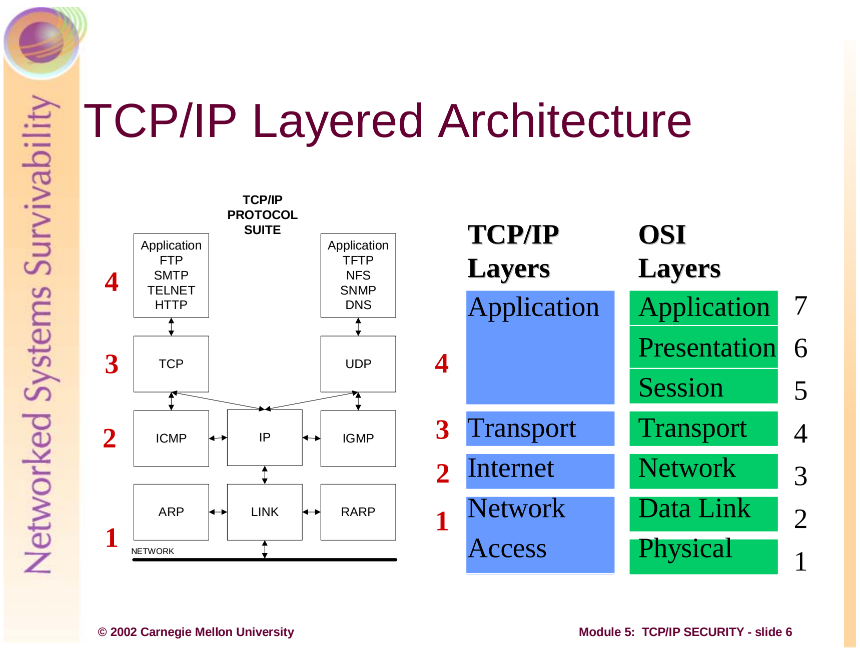
Remote login and remote file transfer were finally implemented in a protocol called the Network Control Program (NCP). Later, electronic mail was added through File Transfer Protocol (FTP). Together with NCP's remote logins and file transfer, this formed the basic services for ARPANET.

By 1973, it was clear that NCP was unable to handle the growing volume of traffic and proposed new functionality. A project was begun to develop a new protocol. The TCP/IP and gateway architectures were first proposed in 1974. The published article by Cerf and Kahn ("A Protocol for Packet Network Interconnection") described a system that provided a standardized application protocol that also used end-to-end acknowledgments. Neither of these concepts were really novel at the time, but more importantly (and with considerable vision), Cerf and Kahn suggested that the new protocol be independent of the underlying network and computer hardware. Also, they proposed universal connectivity throughout the network. These two ideas were radical in a world of proprietary hardware and software, because they would enable any kind of platform to participate in the network. The protocol was developed and became known as TCP/IP.

A series of RFCs (Requests for Comment, part of the process for adopting new Internet Standards) was issued in 1981, standardizing TCP/IP version 4 for the ARPANET. In 1982, TCP/IP replaced NCP as the dominant protocol of the growing network, which was now connecting machines across the continent. It is estimated that a new computer was connected to ARPANET every 20 days during its first decade. (That might not seem like much compared to the current estimate of the Internet's size doubling every year, but in the early 1980s it was a phenomenal growth rate.)

During the development of ARPANET, it became obvious that nonmilitary researchers could use the network to their advantage, enabling faster communication of ideas as well as faster physical data transfer. A proposal to the National Science Foundation led to funding for the Computer Science Network in 1981, joining the military with educational and research institutes to refine the network. This led to the splitting of the network into two different networks in 1984. MILNET was dedicated to unclassified military traffic, whereas ARPANET was left for research and other nonmilitary purposes. ARPANET's growth and subsequent demise came with the approval for the Office of Advanced Scientific Computing to develop wide access to supercomputers. The Department of Defense finally declared ARPANET obsolete in 1990, when it was officially dismantled.

The Internet has grown far beyond its original scope. The original networks and agencies that built the Internet no longer play a crucial role for the current Internet. The Internet has evolved from a simple backbone network, through a three-tiered hierarchical structure, to a huge network of global networks. Through all of this incredible change one thing has remained constant: the Internet is built on the TCP/IP protocol suite. [ISOC 2001]



Networked Systems Survivability

TCP/IP Layered Architecture

	TCP/IP Layers	OSI Layers	
4	Application	Application	7
		Presentation	6
		Session	5
3	Transport	Transport	4
2	Internet	Network	3
1	Network Access	Data Link	2
		Physical	1

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 6

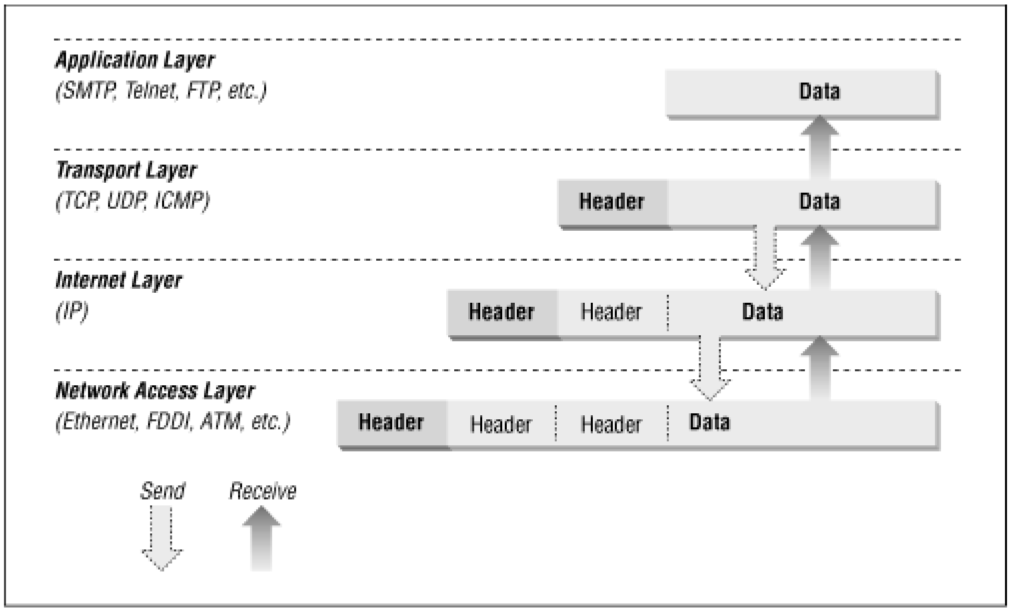
The OSI Reference Model contains seven *layers* that define the functions of data communications protocols. Each layer of the OSI model represents a task performed when data is transferred between cooperating applications across a network. Looking at this slide, you can see that the protocols are like a pile of building blocks stacked one upon another. Because of this appearance, the structure is often called a *stack* or *protocol stack*. A layer does not define a single protocol; it defines a data communications function that may be performed by any number of protocols. Therefore, each layer may contain multiple protocols, each providing a service suitable to the function of that layer. For example, the Hypertext Transfer Protocol and the Simple Network Management Protocol both provide user services and both are part of the Application Layer. Every protocol communicates with its peer. A *peer* is an implementation of the same protocol in the equivalent layer on a remote system (i.e., the local file transfer protocol is the peer of a remote file transfer protocol). Peer level communications must be standardized for successful communications to take place. In the abstract, each protocol is only concerned with communicating to its peer; it does not care about the layer above or below it.

There must also be conformity on how to pass data between the layers on a single computer, because every layer is involved in sending data from a local application to an equivalent remote application. The upper layers rely on the lower layers to transfer the data over the underlying network. Data is passed down the stack from one layer to the next, until it is transmitted over the network by the Physical Layer protocols. At the remote end, the data is passed up the stack to the receiving application. The individual layers do not need to know how the layers above and below them function; they only need to know how to pass data to them. Isolating network communications functions in different layers minimizes the impact of technological change on the entire protocol suite. New applications can be added without changing the physical network, and new network hardware can be installed without rewriting the application software.

Although the OSI model is useful, the TCP/IP protocols don't match its structure exactly. Generally, TCP/IP is described using three to five functional layers. The common *DoD reference model*, which is also known as the *Internet reference model* is shown in the slide.

As in the OSI model, data is passed down the stack when it is being sent to the network, and up the stack when it is being received from the network. The four-layered structure of TCP/IP is seen in the way data is handled as it passes down the protocol stack from the Application Layer to the underlying physical network. Each layer in the stack adds control information to ensure proper delivery. This control information is called a *header* because it is placed in front of the data to be transmitted. Each layer treats all of the information it receives from the layer above as data and places its own header in front of that information. The addition of delivery information at every layer is called *encapsulation*. (Figure 1 illustrates this.) When data is received, the opposite happens. Each layer strips off its header before passing the data on to the layer above. As information flows back up the stack, information received from a lower layer is interpreted as both a header and data. [Cisco 1]

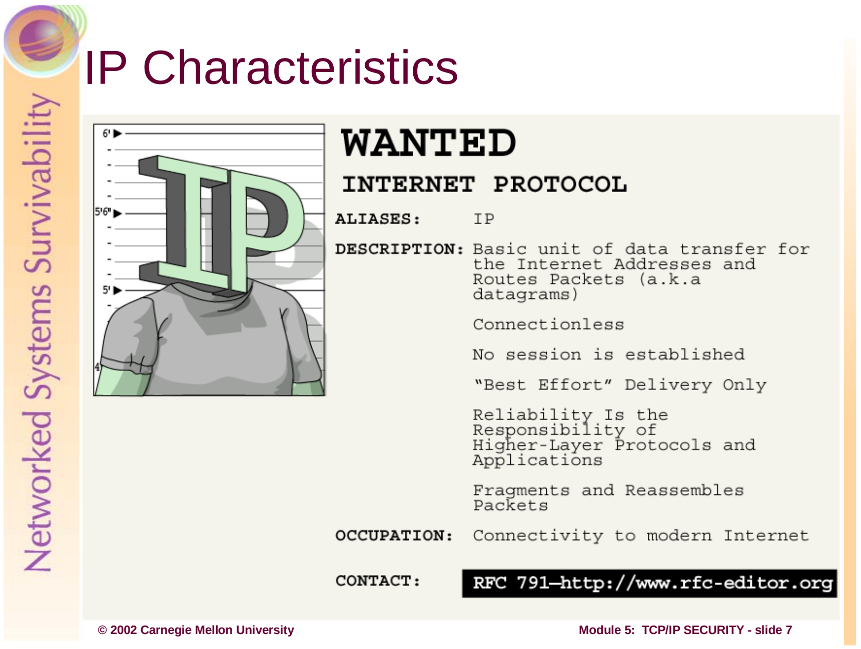
Figure 1: TCP/IP Encapsulation



An audiovisual web tutorial on TCP/IP is available at:

http://www.rad.com/audio_presentations/tcp_ip/slide01.htm

The cyber-instructor has a cool British accent too. Check it out!


Networked Systems Survivability
6'
5'6"
5'
4'

IP Characteristics

WANTED

INTERNET PROTOCOL

ALIASES:	IP
DESCRIPTION:	Basic unit of data transfer for the Internet Addresses and Routes Packets (a.k.a datagrams)
	Connectionless
	No session is established
	"Best Effort" Delivery Only
	Reliability Is the Responsibility of Higher-Layer Protocols and Applications
	Fragments and Reassembles Packets
OCCUPATION:	Connectivity to modern Internet
CONTACT:	RFC 791–http://www.rfc-editor.org

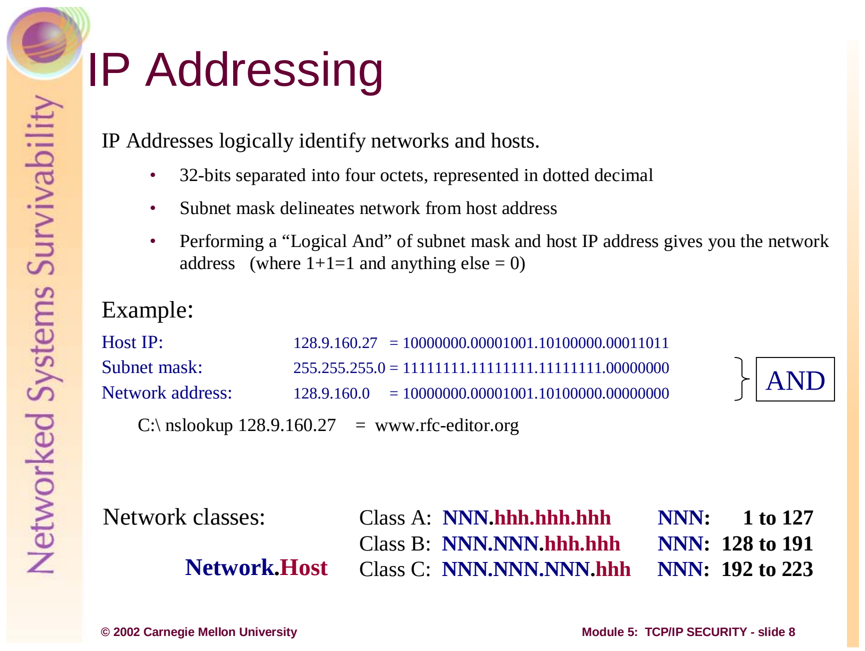
© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 7

The Internet Protocol is the building block of the Internet. Its functions include:

- Defining the packet (or datagram), which is the basic unit of transmission in the Internet
- Defining the Internet addressing scheme
- Moving data between the Network Access Layer and the Transport Layer
- Routing packets to remote hosts
- Performing fragmentation and re-assembly of packets

As stated in the slide, IP is a *connectionless protocol*. This means that IP does not exchange any control information to establish an end-to-end connection before transmitting data. In contrast, a *connection-oriented protocol* exchanges control information with the remote system to verify that both systems are ready to transmit data before any data is sent. This is sometimes referred to as a handshake. When the handshaking is successful, the systems have established a *connection*. Internet Protocol relies on protocols in other layers (transport or application) to establish the connection if they require connection-oriented service.

IP also relies on protocols in the other layers to provide error detection and error recovery. The Internet Protocol is sometimes called an *unreliable protocol* because it contains only minimal error detection and recovery code—a rudimentary checksum is performed only on the IP Header. This is not to say that the protocol is prone to errors and poor performance. IP can be relied upon to accurately deliver data to the destination network, but it doesn't check whether that data was correctly received. Protocols in other layers of the TCP/IP architecture provide this checking when it is required. [Kessler 2001]


Networked Systems Survivability

IP Addressing

IP Addresses logically identify networks and hosts.

- 32-bits separated into four octets, represented in dotted decimal
- Subnet mask delineates network from host address
- Performing a "Logical And" of subnet mask and host IP address gives you the network address (where 1+1=1 and anything else = 0)

Example:

Host IP:	128.9.160.27 = 10000000.00001001.10100000.00011011	AND
Subnet mask:	255.255.255.0 = 11111111.11111111.11111111.00000000	
Network address:	128.9.160.0 = 10000000.00001001.10100000.00000000	

C:\ nslookup 128.9.160.27 = www.rfc-editor.org

Network classes:

Network.Host

Class A:	**NNN.hhh.hhh.hhh**	**NNN: 1 to 127**
Class B:	**NNN.NNN.hhh.hhh**	**NNN: 128 to 191**
Class C:	**NNN.NNN.NNN.hhh**	**NNN: 192 to 223**

© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 8

An IP address consists of four octets (1 octet = 8 bits), or 32 bits. The value in each octet ranges from 0 to 255 decimal, or 00000000 - 11111111 binary. Here's how binary octets convert to decimal:

1	1	1	1	1	1	1	1
128	64	32	16	8	4	2	1

(128+64+32+16+8+4+2+1=255)

Now here's a sample octet conversion:

0	1	0	0	0	0	0	1
0	64	0	0	0	0	0	1

(0+64+0+0+0+0+0+1=65)

And this is a sample address representation (4 octets):

10.	1.	23.	19	(decimal)
00001010.	00000001.	00010111.	00010011	(binary)

These octets are broken down to provide an addressing scheme that can accommodate large and small networks. There are five different classes of networks, A to E. Here, we'll only be addressing classes A to C, since classes D and E are reserved. To determine the class of an address, look at the first octet of the dotted-decimal address.

Class A: 1 - 126	(eg. 10.1.23.19)
Class B: 128-191	(eg. 172.16.19.48)
Class C: 192-223	(eg. 193.18.9.10)

In a class A address, the first octet is the network portion, so the class A example above has a major network address of 10. Octets 2, 3, and 4 (the next 24 bits) are for the network manager to divide into subnets and hosts as she sees fit. Class A addresses are used for networks that have more than 65,536 hosts (actually, up to 16,581,375 hosts!). In a class B address, the first two octets are the network portion, so the class B example above has a major network address of 172.16. Octets 3 and 4 (16 bits) are for local subnets and hosts. Class B addresses are used for networks that have between 256 and 65,536 hosts.

In a class C address, the first three octets are the network portion. The class C example above has a major network address of 193.18.9. Octet 4 (8 bits) is for local subnets and hosts - perfect for networks with less than 256 hosts.

In order to use your addresses, you need to understand subnetting. Subnetting allows you to create multiple logical networks that exist within a single Class A, B, or C network. If you don't subnet, you will only be able to use one network from your Class A, B, or C network. Unless you have been assigned many major networks, you really need to subnet.

A subnet mask is defined for each IP address. The subnet mask identifies which portion of the 4 octets is used to identify the network, with the remaining bits identifying the node. If you want no subnetting, use these default masks (255 - strictly follow number, 0 - wildcard):

Class A: 255.0.0.0
Class B: 255.255.0.0
Class C: 255.255.255.0

Let's use these two addresses for some examples: 171.68.3.3 and 171.68.2.3. If the subnet mask is 255.255.255.0, the first 24 bits are masked, so a router compares the first 3 octets of the two addresses. Since the masked bits are not the same, the router knows that these addresses belong to different subnets.

If the subnet mask is 255.255.0.0, the first 16 bits are masked, so a router compares the first 2 octets of the two addresses. Since the masked bits are the same, the router knows that these addresses belong to the same subnet. Nodes and routers use the mask to identify the network on which an address resides. For instance, imagine that Pittsburgh proper is a class B network, and think of the streets as subnets. Each street must have a unique name. How would the postal service deliver a letter or find the correct destination if there were two Craig Streets? Each house number can be thought of as a unique identifier for that street. The house numbers themselves can be duplicated on other streets: 33 Market Street is not the same as 33 Craig Street.

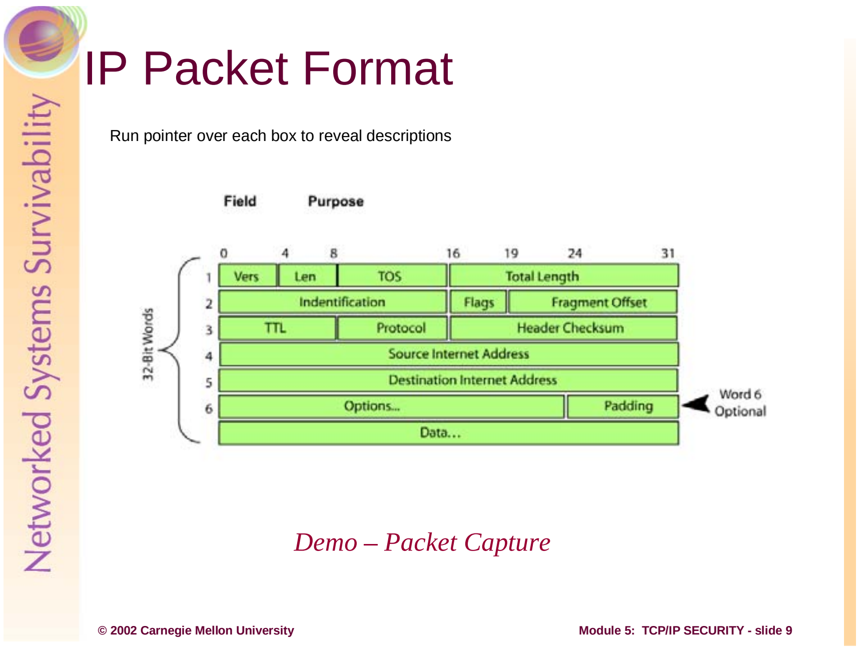
Pittsburgh.Craig.33	Pittsburgh.Market.33
171.68. 3. 3	171.68. 2. 3

You need to perform a logical "AND" operation to find out what subnet your node is in. Performing an "AND" operation means that anytime you "AND" a 0 value to another 0 or a 1 value, the result is 0. Only a 1 ANDed with another 1 value will result in a 1 value. Here's how it works:

0 AND 0 = 0,   0 AND 1 = 0,   1 AND 1 = 1

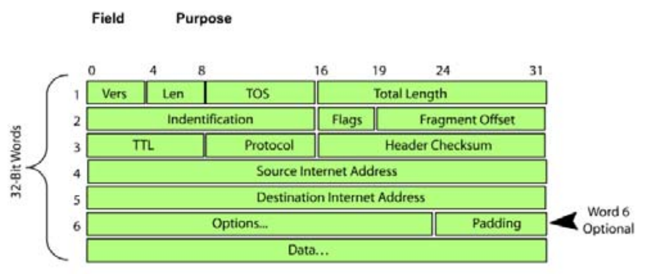
Let's compare our sample addresses (171.68.3.3 and 171.68.2.3) against the subnet mask 255.255.240.0. We need to compare the binary representation of the third octet of the mask with the binary representation of the third octets of the addresses. In order to do this, we'll perform a logical "AND" operation on the corresponding bits in each octet.

The masked bits are those that are "turned on," or 1 in the mask. Since the masked bits in both addresses are the same, the router knows that these addresses belong to the same subnet.



IP Packet Format

Run pointer over each box to reveal descriptions



Demo – Packet Capture

© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 9

The basic IP packet header format is shown in the slide above. The format of the diagram is consistent with the RFC; bits are numbered from left-to-right, starting at 0. Each row represents a single *32-bit word*; note that an IP header will be at least 5 words (20 bytes) in length. The fields contained in the header, and their functions, are:

Version: Specifies the IP version of the packet. The current version of IP is version 4, so this field will contain the binary value 0100. [NOTE: Actually, many IP version numbers have been assigned besides 4 and 6; see the IANA's list of IP Version Numbers.]

Internet Header Length (IHL): Indicates the length of the packet header in 32 bit (4 octet) words. A minimum-length header is five 32-bit words, so this field always has a value of at least 5 (0101) Since the maximum value of this field is 15, the IP Header can be no longer than fifteen 32-bit words.

Type of Service (TOS): Allows an originating host to request different classes of service for packets it transmits. Although not generally supported today in IPv4, the TOS field can be set by the originating host in response to service requests across the Transport Layer/Internet Layer service interface, and can specify a service priority (0-7) or can request that the route be optimized for either cost, delay, throughput, or reliability.

Total Length: Indicates the length (in bytes, or octets) of the entire packet, including both header and data. Given the size of this field, the maximum size of an IP packet is 64 KB, or 65,535 bytes. In practice, packet sizes are limited to the maximum transmission unit (MTU).

Identification: Used when a packet is fragmented into smaller pieces while traversing the Internet, this identifier is assigned by the transmitting host so that different fragments arriving at the destination can be associated with each other for reassembly.

Flags: Also used for fragmentation and reassembly. The first bit is called the More Fragments (MF) bit, and is used to indicate the last fragment of a packet so that the receiver knows that the packet can be reassembled. The second bit is the Don't Fragment (DF) bit, which suppresses fragmentation. The third bit is unused (and always set to 0).

Fragment Offset: Indicates the position of this fragment in the original packet. In the first packet of a fragment stream, the offset will be 0; in subsequent fragments, this field will indicates the offset in increments of 8 bytes.

Time-to-Live (TTL): A value from 0 to 255, indicating the number of hops that this packet is allowed to take before discarded within the network. Every router that sees this packet will decrement the TTL value by one; if it gets to 0, the packet will be discarded.

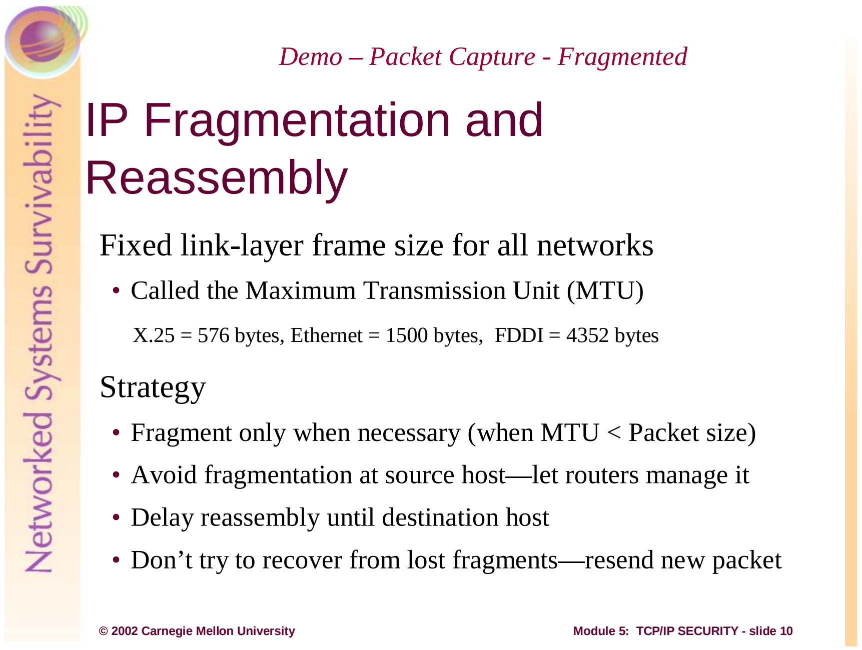
Protocol: Indicates the higher layer protocol contents of the data carried in the packet; options include ICMP (1), TCP (6), UDP (17), or OSPF (89). A complete list of IP protocol numbers can be found at the IANA's list of Protocol Numbers. An implementation-specific list of supported protocols can be found in the `protocol` file, generally found in the `/etc` (Linux/Unix), `c:\windows` (Windows 9x, ME), or `c:\winnt\system32\drivers\etc` (Windows NT, 2000) directory.

Header Checksum: Carries information to ensure that the received IP header is error-free. Remember that IP provides an unreliable service and, therefore, this field only checks the header rather than the entire packet. The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero. This means that the header is chunked into (a maximum of 12) 16 bit pieces. These pieces are then summed using one's complement math. Finally, a one's complement is calculated for this 16 bit value—which is the actual 16 bit checksum.

Source Address: IP address of the host sending the packet.

Destination Address: IP address of the host intended to receive the packet.

Options: A set of options which may be applied to any given packet, such as sender-specified source routing or security indication. The option list may use up to 40 bytes (10 words), and will be padded to a word boundary; IP options are taken from the IANA's list of IP Option Numbers.



Demo – Packet Capture - Fragmented

IP Fragmentation and Reassembly

Fixed link-layer frame size for all networks

- Called the Maximum Transmission Unit (MTU)

  X.25 = 576 bytes, Ethernet = 1500 bytes, FDDI = 4352 bytes

Strategy

- Fragment only when necessary (when MTU < Packet size)
- Avoid fragmentation at source host—let routers manage it
- Delay reassembly until destination host
- Don't try to recover from lost fragments—resend new packet

Networked Systems Survivability

© 2002 Carnegie Mellon University Module 5: TCP/IP SECURITY - slide 10

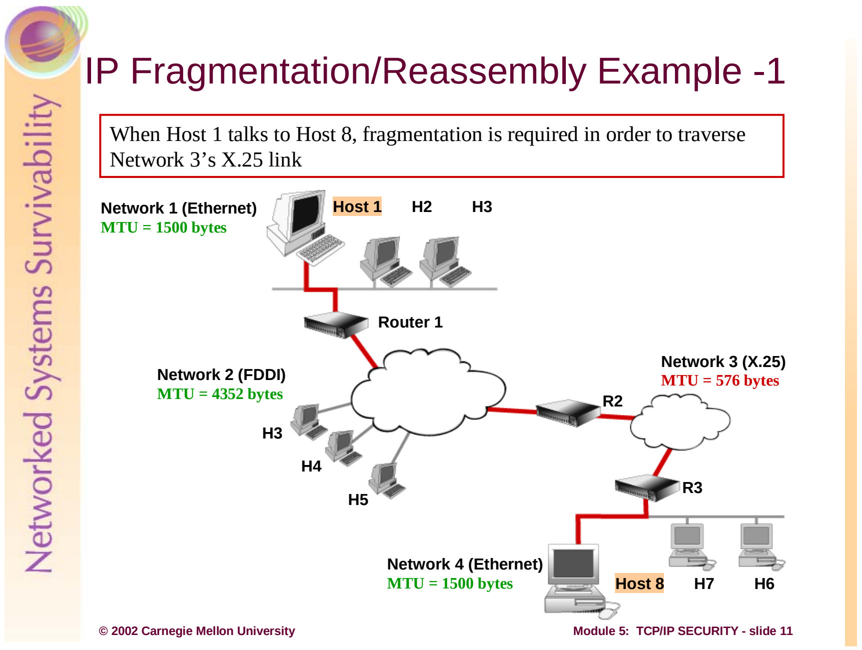
Whenever the IP layer receives an IP packet to send, it determines which interface on the local router the packet is being sent out on and then queries that interface to obtain its MTU. IP compares the MTU with the packet size and performs fragmentation, if necessary. Fragmentation can take place either at the original sending host or at an intermediate router—although source hosts typically do this for testing or diagnostic purposes and generally not for operational communication.

When an IP packet is fragmented, it is not reassembled (typically) until it reaches its final destination. The IP layer at the destination system performs the reassembly. The goal is to make fragmentation and reassembly transparent to the transport layer (TCP and UDP), which it is, except for possible performance degradation. It is also possible for the fragment of a packet to again be fragmented (possibly more than once). The information maintained in the IP header for fragmentation and reassembly provides enough information to do this.

Recalling the IP header, the following fields are used in fragmentation. The *identification* field contains a unique value for each IP packet that the sender transmits. This number is copied into each fragment of a particular packet. The flags field uses one bit as the "more fragments" bit. This bit is turned on (is a 1) for each fragment comprising a packet except the final fragment. The fragment offset field contains the offset of this fragment from the beginning of the original packet. Thus providing a pointer as to the position the fragment resides in original packet. Also, when a packet is fragmented the *total length* field of each fragment is changed to be the size of that fragment.

Finally, one of the bits in the flags field is called the "don't fragment" bit. If this is turned on, IP will not fragment the packet. In this case, if a “don’t fragment” packet encounters a network that it cannot traverse (because of a smaller MTU) it will be discarded. The router that discarded the packet will send an ICMP error message (fragmentation needed but don't fragment bit set) back to the originator. When an IP packet is fragmented, each fragment becomes its own packet, with its own IP header, and is routed independently of any other packets. This makes it possible for the fragments of a packet to arrive at the final destination out of order, but there is enough information in the IP header to allow the receiver to reassemble the fragments correctly.

Although IP fragmentation looks transparent, there is one facet that makes it somewhat inefficient: if one fragment is lost the entire packet must be retransmitted. To understand why this happens, realize that IP itself has no timeout and retransmission facility built in--that is the responsibility of higher layer protocols. TCP performs timeout and retransmission although UDP does not. Some UDP applications perform timeout and retransmission for themselves. When a fragment is lost that came from a TCP segment, TCP will time out and retransmit the entire TCP segment, which corresponds to a new IP packet. There is no way to resend only one fragment of a packet. Indeed, if the fragmentation was done by an intermediate router, (and not the originating host), there is no way for the originating host to know how the packet was fragmented.



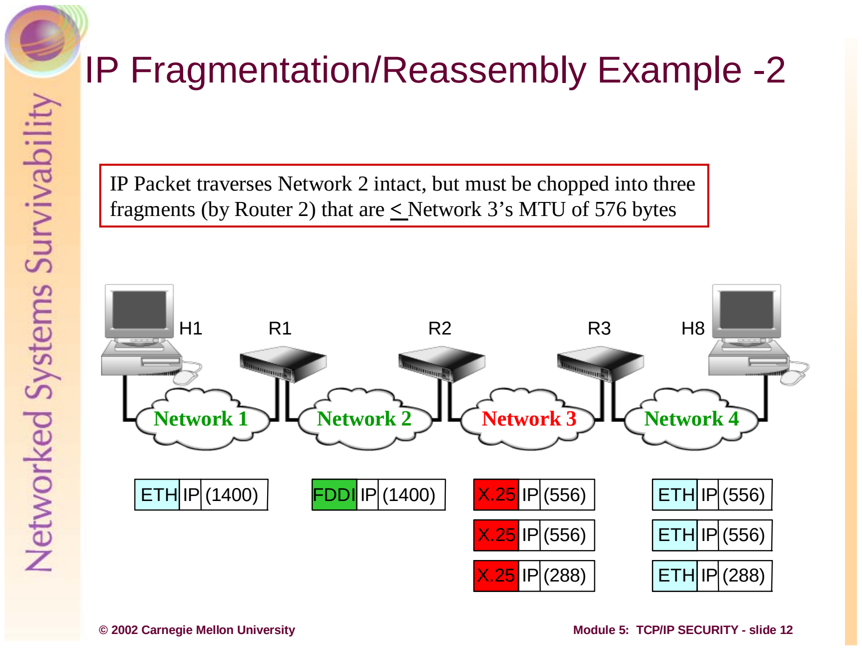
IP Fragmentation/Reassembly Example -1

When Host 1 talks to Host 8, fragmentation is required in order to traverse Network 3's X.25 link

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 11

The sample internetwork in this slide accurately depicts the need for IP Fragmentation. X.25 networks are still very common throughout Europe and other parts of the world, therefore this type of fragmentation scenario is played out millions of times each day.



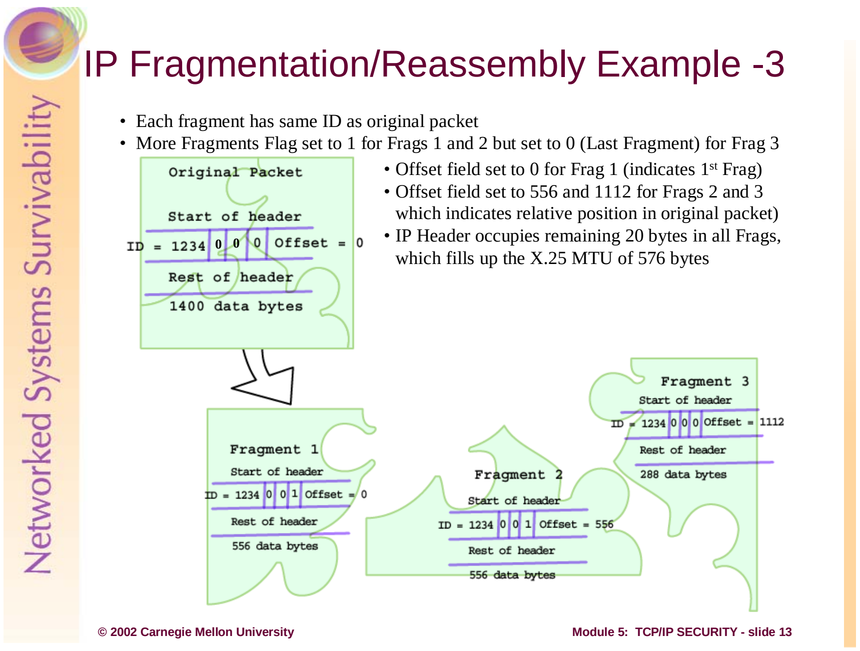
IP Fragmentation/Reassembly Example -2

IP Packet traverses Network 2 intact, but must be chopped into three fragments (by Router 2) that are ≤ Network 3's MTU of 576 bytes

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 12

The packet from Host 1 reaches Router 1 which proceeds to strip off the Ethernet frame and encapsulate the IP packet within a FDDI frame for transport onto Network 2. Router 2 then strips off the FDDI frame, reads the packet's destination IP address (for Host 8) and determines that Network 3 is the correct (and in this case only) route. Because the packet is larger than Network 3's MTU, the packet is fragmented. Finally, Router 3 strips off the X.25 frames, consults its routing table, and then encapsulates the fragments within Ethernet frames for transport on Network 4. Notice that Router 3 does not attempt to reassemble the fragments into the original packet—that is left up to the destination host.



IP Fragmentation/Reassembly Example -3

- Each fragment has same ID as original packet
- More Fragments Flag set to 1 for Frags 1 and 2 but set to 0 (Last Fragment) for Frag 3
- Offset field set to 0 for Frag 1 (indicates 1st Frag)
- Offset field set to 556 and 1112 for Frags 2 and 3 which indicates relative position in original packet)
- IP Header occupies remaining 20 bytes in all Frags, which fills up the X.25 MTU of 576 bytes

© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 13

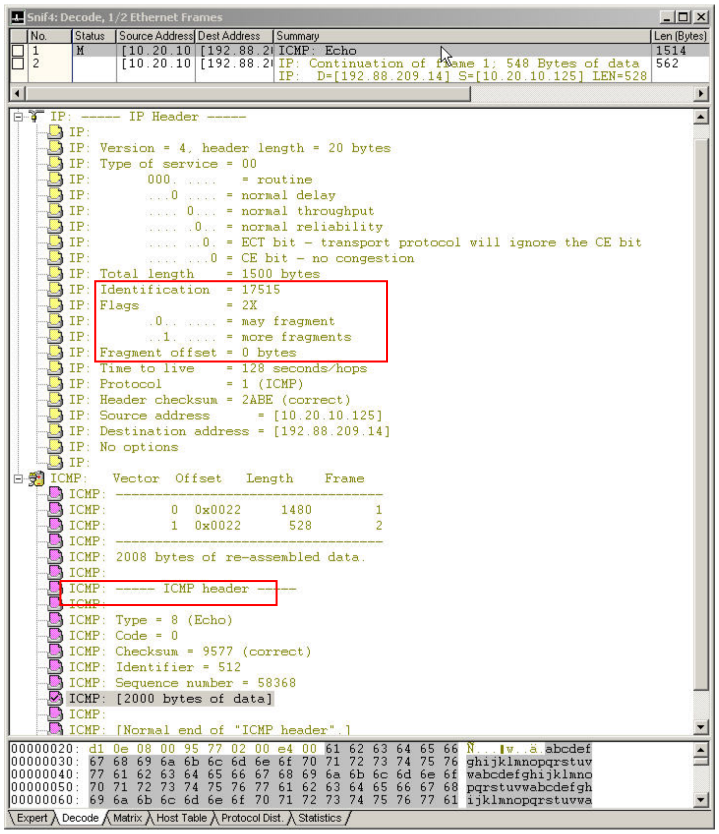
Consider the following IP packet decode:

We used the ping –l command, to send ICMP Echo packets that were larger than our MTU of 1500 bytes (Ethernet)—in this case 2000 bytes.

As a result, our local host's IP Stack was forced to Fragment the Echo packets in order to route them successfully to our destination host.

Notice in the IP header that the MF (more fragments) flag is set to 1, indicating that Fragmentation has occurred. All fragments will have the same Identification (17515), as well as the same Source and Destination address.

The Fragment Offset field for the next fragment will indicate the position (in bytes) in the original packet.



Snif4: Decode, 1/2 Ethernet Frames

No.	Status	Source Address	Dest Address	Summary	Len (Bytes)
1	M	[10.20.10	[192.88.2	ICMP: Echo	1514
2		[10.20.10	[192.88.2	IP: Continuation of frame 1; 548 Bytes of data IP: D=[192.88.209.14] S=[10.20.10.125] LEN=528	562

```
IP: ----- IP Header -----
  IP:
  IP: Version = 4, header length = 20 bytes
  IP: Type of service = 00
  IP:       000. ....   = routine
  IP:       ...0 .... = normal delay
  IP:       .... 0... = normal throughput
  IP:       .... .0.. = normal reliability
  IP:       .... ..0. = ECT bit - transport protocol will ignore the CE bit
  IP:       .... ...0 = CE bit - no congestion
  IP: Total length    = 1500 bytes
  IP: Identification  = 17515
  IP: Flags           = 2X
  IP:       .0.. .... = may fragment
  IP:       ..1. .... = more fragments
  IP: Fragment offset = 0 bytes
  IP: Time to live    = 128 seconds/hops
  IP: Protocol        = 1 (ICMP)
  IP: Header checksum = 2ABE (correct)
  IP: Source address      = [10.20.10.125]
  IP: Destination address = [192.88.209.14]
  IP: No options
  IP:
ICMP:   Vector  Offset   Length    Frame
  ICMP: ----------------------------------
  ICMP:       0  0x0022     1480        1
  ICMP:       1  0x0022      528        2
  ICMP: ----------------------------------
  ICMP: 2008 bytes of re-assembled data.
  ICMP:
  ICMP: ----- ICMP header -----
  ICMP:
  ICMP: Type = 8 (Echo)
  ICMP: Code = 0
  ICMP: Checksum = 9577 (correct)
  ICMP: Identifier = 512
  ICMP: Sequence number = 58368
  ICMP: [2000 bytes of data]
  ICMP:
  ICMP: [Normal end of "ICMP header".]
```

```
00000020: d1 0e 08 00 95 77 02 00 e4 00 61 62 63 64 65 66 Ñ....w..ä.abcdef
00000030: 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 ghijklmnopqrstuv
00000040: 77 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f wabcdefghijklmno
00000050: 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 pqrstuvwabcdefgh
00000060: 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 61 ijklmnopqrstuvwa
```

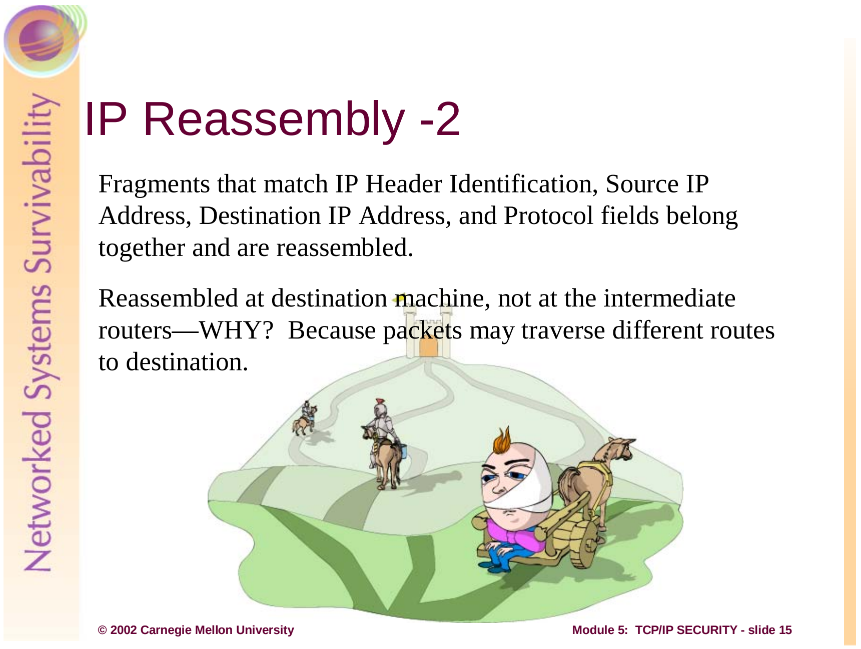
Networked Systems Survivability

IP Reassembly -1

ONE BUCKET GOES DOWN THE LEFT PATH, ONE DOWN THE MIDDLE AND THE "EGG-MAN" TO THE RIGHT.

© 2002 Carnegie Mellon University — Module 5: TCP/IP SECURITY - slide 14

This slide shows how fragments can take different routes to the destination. Although the specification recommends that fragments be reassembled at the final destination, some firewalls and intrusion detection systems do either complete or virtual reassembly prior to forwarding them into the protected network. This technique can be difficult to implement because of routing changes and protocol requirements. It also can potentially open DOS vulnerabilities on the firewall itself—covered later in this workbook. Many applications these days attempt to avoid fragmentation altogether by simply using the Do Not Fragment flag in the IP header. With these applications, it is typical to initially disallow fragmentation and then adjust if ICMP Type 3 Code 4 Messages are received (Destination Unreachable--Fragmentation needed and DF flag set).



Networked Systems Survivability

IP Reassembly -2

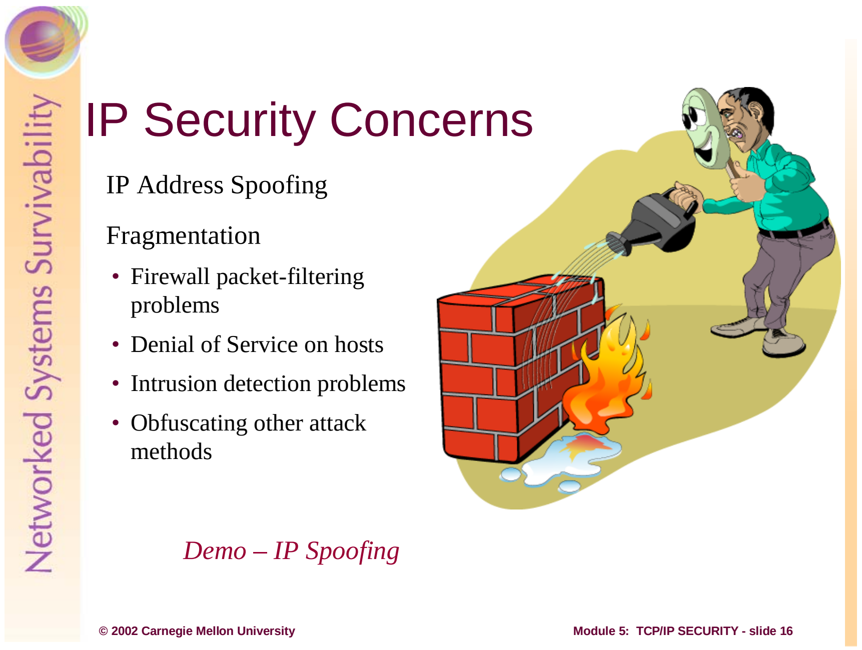
Fragments that match IP Header Identification, Source IP Address, Destination IP Address, and Protocol fields belong together and are reassembled.

Reassembled at destination machine, not at the intermediate routers—WHY? Because packets may traverse different routes to destination.

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 15

In our example, Host 8 checks all three fragments to make sure the appropriate IP Header fields are identical (see first bullet of slide), and then reassembles the fragments into the original packet. It uses the Fragment Offset field to put it together in the correct order.



Networked Systems Survivability

IP Security Concerns

IP Address Spoofing

Fragmentation

- Firewall packet-filtering problems
- Denial of Service on hosts
- Intrusion detection problems
- Obfuscating other attack methods

Demo – IP Spoofing

© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 16

IP Spoofing is a technique whereby intruders pretend to be sending data from an IP address other than their own. The IP layer assumes that the source address on any IP packet it receives is the same IP address as the system that actually sent the packet -- it does no authentication. Many higher level protocols and applications also make this assumption, so it seems that anyone able to forge the source address of an IP packet (called "spoofing" an address) could potentially get unauthorized privileges.

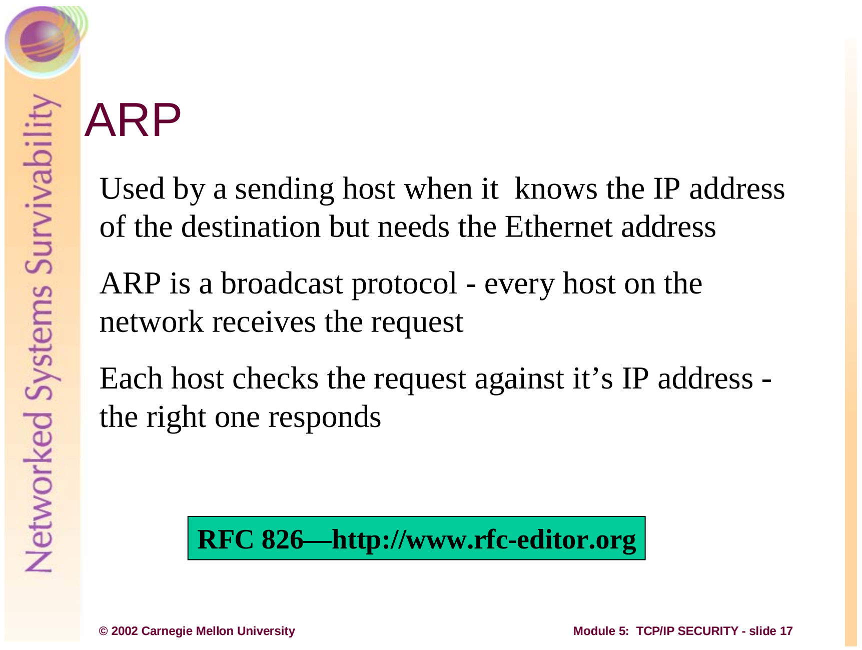
However, there are two catches. The first catch is that communication is likely to be one-way. The remote host will send all replies to the spoofed source address -- not to the host actually doing the spoofing. So, an attacker using IP spoofing is unlikely to see output from the remote system (unless they have some other method of eavesdropping on the network between the other two hosts). The second catch is that an attacker needs to use the correct TCP sequence numbers if they plan on establishing a TCP connection with the attacked host (most common services, like Telnet, FTP, and r-commands use TCP). The final ACK in a three-way handshake (see slide 27) must contain the other host's initial sequence number (ISN), otherwise the connection cannot complete; because the ISN in the SYN+ACK packet is sent to the real host, an attacker must get this ISN by some other method. If the attacker could eavesdrop on the packets sent from the other host, he could see the ISN. Similarly, if the attacker was unable to eavesdrop, but could somehow guess the other host's ISN, he can complete the connection and conduct a one way conversation (this may be sufficient to initiate some other form of two-way communication). Unfortunately for the TCP/IP community, methods to overcome both challenges in IP Spoofing have been developed.

Some attacks manipulate the fragmentation feature of IP. Many firewalls including industry leader Checkpoint Firewall-1, actually reassemble (they call it virtual packet-reassembly) all fragmented datagrams so that the entire IP packet can be tested against its access rules before it is forwarded into the protected network. As a result, if a firewall receives an IP fragment (IP Flag MF=1), it will hold this in its memory buffer (or queue), waiting for all remaining fragments. Therefore, all a hacker need do is flood the firewall with IP fragments and before long it will bog down and no longer be able to pass legitimate packets. Some firewalls, like IP Tables in Linux, have tuned this operation to survive some flooding. This feature is enabled through the CONFIG _IP _ALWAYS _DEFRAG option in the Linux kernel, however it does impart a significant performance burden on the system and potentially the protected network depending on its available bandwidth. Checkpoint has since provided a patch for the vulnerability described, but as is often the case, many operational systems are inadequately patched.

A different DOS attack utilizing IP fragmentation is the Teardrop attack. This attack makes each fragment look like the original IP packet except that it contains an offset field that says, for instance, "This fragment is carrying bytes 600 through 800 of the original (non-fragmented) IP packet." The Teardrop program creates a series of IP fragments with overlapping fragment offset fields. When these fragments are reassembled at the destination host, some systems will crash, hang, or reboot.

Because of the intricacies of IP fragmentation, many intrusion detection systems have difficulty identifying malicious packet streams. Like firewalls, IDS need to inspect a complete (non-fragmented packet) so as to be accurate in comparing packets against their signature databases. Fragmentation is used to hide or obfuscate malicious attacks and as a means of eluding access controls and detection.

So as you can see, IP fragmentation can be rather challenging for security professionals. Some administrators disallow fragmented packets into their networks completely—which may cause access and availability problems. IP fragmentation has been termed a "hackers dream" and a security officer's nightmare—the truth is probably somewhere in the middle for both.



ARP

Used by a sending host when it knows the IP address of the destination but needs the Ethernet address

ARP is a broadcast protocol - every host on the network receives the request

Each host checks the request against it's IP address - the right one responds

RFC 826—http://www.rfc-editor.org

Networked Systems Survivability

© 2002 Carnegie Mellon University

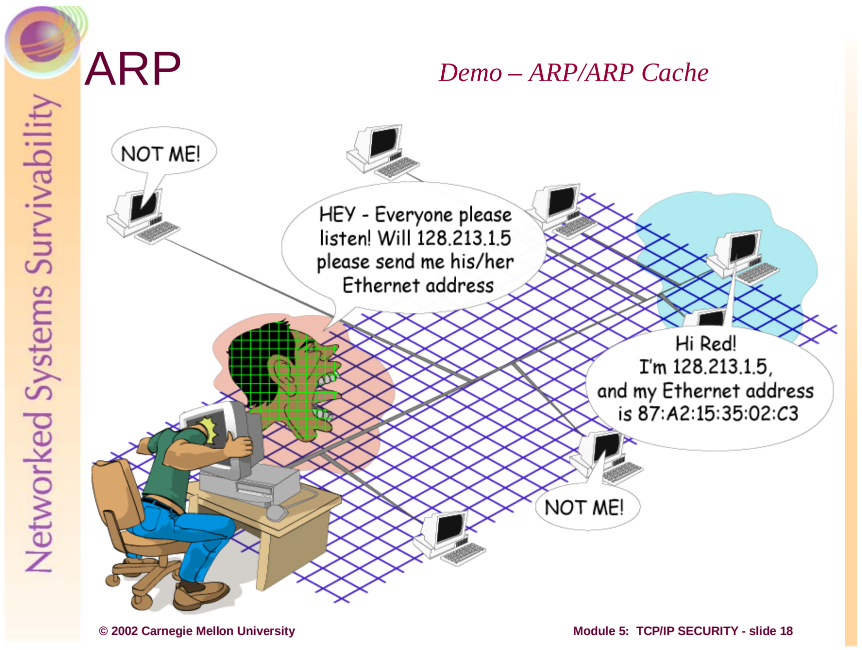
Module 5: TCP/IP SECURITY - slide 17

Address Resolution Protocol (ARP) is a protocol for mapping an Internet Protocol address (IP address) to a physical machine address that is recognized in the local network. For example, in IP Version 4, an address is 32 bits long. In an Ethernet local area network, however, addresses (a.k.a. MAC addresses) for attached devices are 48 bits long. A table, usually called the ARP cache, is used to maintain a correlation between each MAC address and its corresponding IP address. ARP provides the protocol rules for making this correlation and providing address conversion in both directions. There is also a Reverse ARP (RARP) for host machines that don't know their IP address. RARP enables them to request their IP address from the router's ARP cache.

How ARP Works:

When an incoming packet destined for a host machine on a particular local area network arrives at a router, the router asks the ARP program to find a physical host or MAC address that matches the IP address. The ARP program looks in the ARP cache and, if it finds the address, provides it so that the packet can be converted to the right packet length and formatted and sent to the machine. If no entry is found for the IP address, ARP broadcasts a request packet in a special format to all the machines on the LAN. A machine that recognizes the IP address as its own replies. ARP updates the ARP cache for future reference and then sends the packet to the MAC address that replied.

Since protocol details differ for each type of local area network, there are separate ARP Requests for Comments (RFC) for Ethernet, ATM, Fiber Distributed-Data Interface, HIPPI, and other protocols.



Networked Systems Survivability

ARP

Demo – ARP/ARP Cache

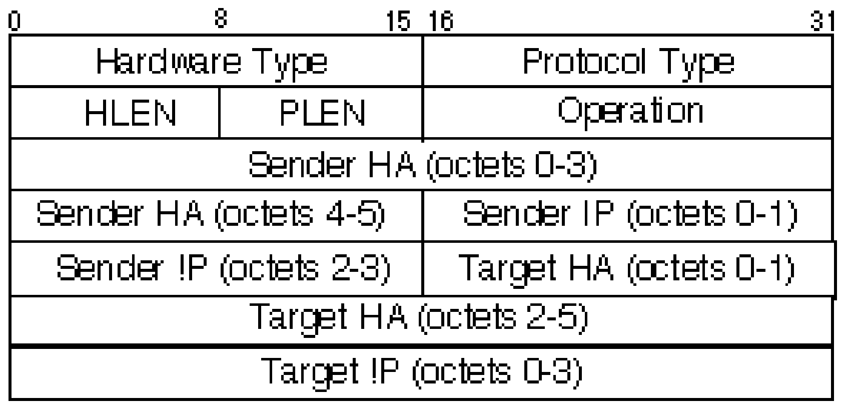
NOT ME!
HEY - Everyone please listen! Will 128.213.1.5 please send me his/her Ethernet address
Hi Red! I'm 128.213.1.5, and my Ethernet address is 87:A2:15:35:02:C3
NOT ME!

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 18

In this example, Host Red sends a broadcast ARP request packet that is processed by every host on the local network. Host Green sends an ARP reply packet directly back (unicast) to Host Red providing its Ethernet address. All other hosts on the network simply drop Host Red's original broadcast packet.

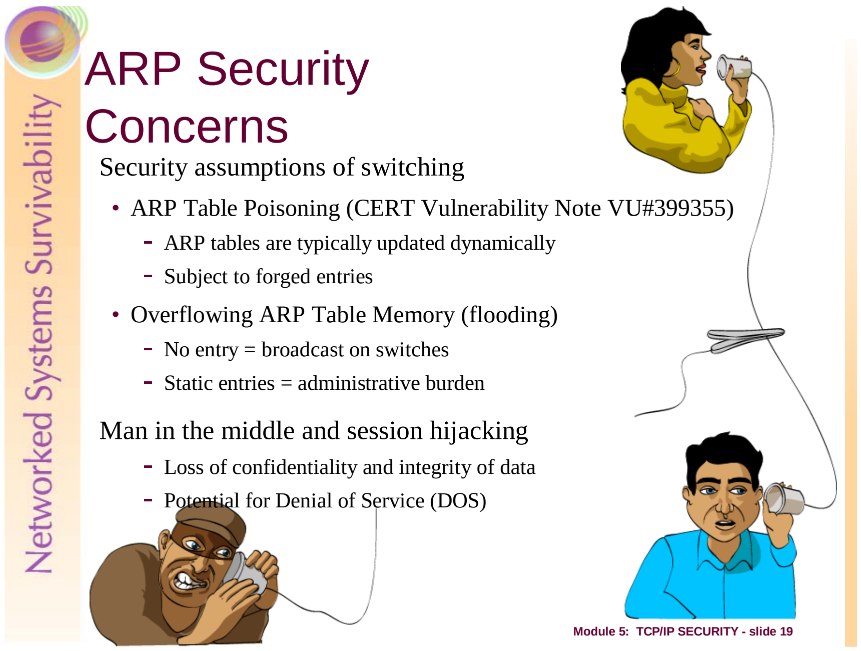
ARP Packet Format:



0 – 7	8 – 15	16 – 31
Hardware Type		Protocol Type
HLEN	PLEN	Operation
Sender HA (octets 0-3)		
Sender HA (octets 4-5)		Sender IP (octets 0-1)
Sender IP (octets 2-3)		Target HA (octets 0-1)
Target HA (octets 2-5)		
Target IP (octets 0-3)		

ARP Parameters (for Ethernet and Ipv4):

Hardware Type	1 = Ethernet	2 = IEEE 802 LAN
Protocol Type	2048 = IPv4 (0x0800)	
Hardware Length (HLEN)	6 = (octets) for Ethernet/IEEE 802 LAN	
Protocol Length (PLEN)	4 = (octets) for IPv4	
Operation:	1 = Request	2 = Reply


Networked Systems Survivability
Module 5: TCP/IP SECURITY - slide 19

ARP Security Concerns

Security assumptions of switching

- ARP Table Poisoning (CERT Vulnerability Note VU#399355)
  - ARP tables are typically updated dynamically
  - Subject to forged entries
- Overflowing ARP Table Memory (flooding)
  - No entry = broadcast on switches
  - Static entries = administrative burden

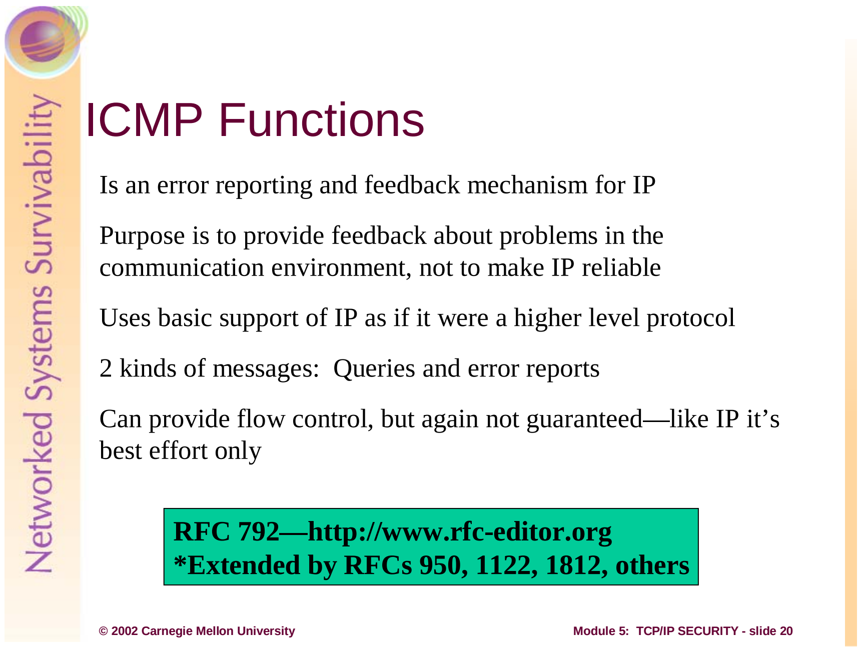
Man in the middle and session hijacking

- Loss of confidentiality and integrity of data
- Potential for Denial of Service (DOS)

The assumption is often made that switches increase security since unicast packets are only seen by the hosts they are destined for. Under normal circumstances this may be true, but there are many cases in which this assumption is faulty. The first instance is on startup or when the switch has not as yet learned the MAC address of a particular host. In this case the packet is typically forwarded out to all interfaces on the switch (broadcast). Until the destination host responds, the switch has not cached the system's MAC address and therefore, doesn't know which port to use to send the packet.

The second instance is when the switch's buffer memory becomes full--due to either too many hosts being attached to the switch (overloading its port density) or by being intentionally filled up as a result of forged ARP packets. If the switch's buffer is completely full, it can no longer write new entries into the ARP cache. In this case, (as mentioned above) switches will typically send packets out every interface. Additionally, a timer is placed on all new entries to the ARP cache, when this timer expires; the entry is deleted from the ARP cache. The next time this machine's MAC address is resolved it will be done so through normal broadcast. It is possible to statically map MAC addresses to physical interfaces within switches, however this can be an administrative burden.

ARP spoofing is even easier (in most cases) than IP spoofing—especially on non-switched networks. Like IP, there is no authentication done by switches, routers, or end hosts. Therefore, it becomes trivial to forge and transmit ARP packets, (for say the MAC address of the default gateway), thereby sending all non-local traffic through the attacker's system. This is commonly referred to as a man-in-the-middle. The intruder can also run a packet capturing program (like sniffer or TCPDump) and learn the IP to MAC correlations for all systems on the local network. With this information, the intruder can spoof the MAC address of any local host, and then potentially receive all traffic intended for the victim. This can result in a DOS for the spoofed host.



Networked Systems Survivability

ICMP Functions

Is an error reporting and feedback mechanism for IP

Purpose is to provide feedback about problems in the communication environment, not to make IP reliable

Uses basic support of IP as if it were a higher level protocol

2 kinds of messages: Queries and error reports

Can provide flow control, but again not guaranteed—like IP it's best effort only

RFC 792—http://www.rfc-editor.org
***Extended by RFCs 950, 1122, 1812, others**

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 20

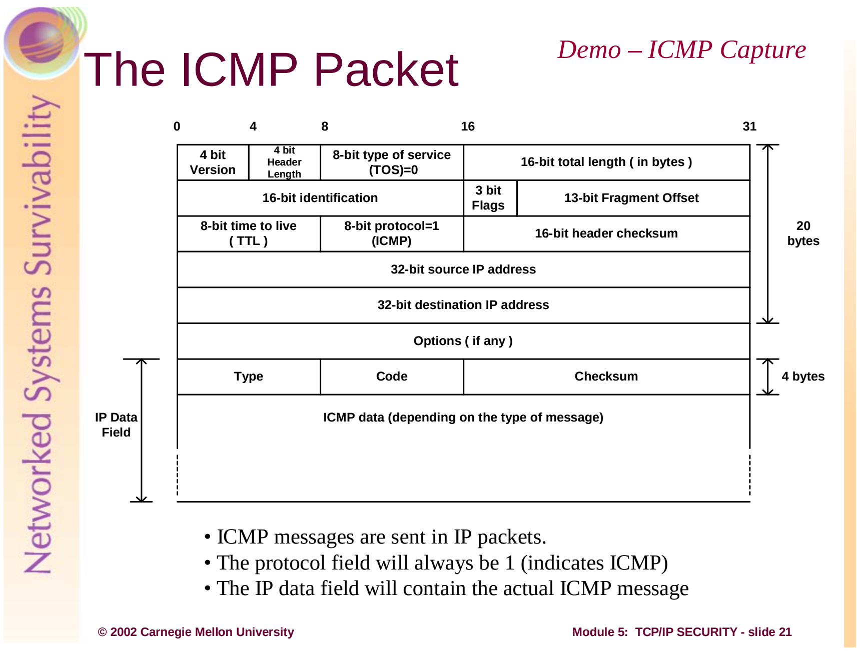
ICMP uses the basic support of IP as if it were a higher level protocol (like TCP or UDP), however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a packet cannot reach its destination, when a router does not have the buffering capacity to forward a packet, and when a router can direct the host to send traffic on a shorter route.

As previously mentioned, the Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a packet will be delivered or a control message will be returned. Some packets may still be undelivered without any report of their loss. The higher level protocols that use IP must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of packets. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. Also ICMP messages are only sent about errors in handling the first fragment of fragmented packets.

For more information, see: http://www.networkcomputing.com/netdesign/1107icmp1.html



The ICMP Packet

- ICMP messages are sent in IP packets.
- The protocol field will always be 1 (indicates ICMP)
- The IP data field will contain the actual ICMP message

© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 21

ICMP messages are sent using the basic IP header. The first octet of the data portion of the packet is an ICMP type field; the value of this field determines the format of the remaining data.

Consider the following ICMP packet decode:

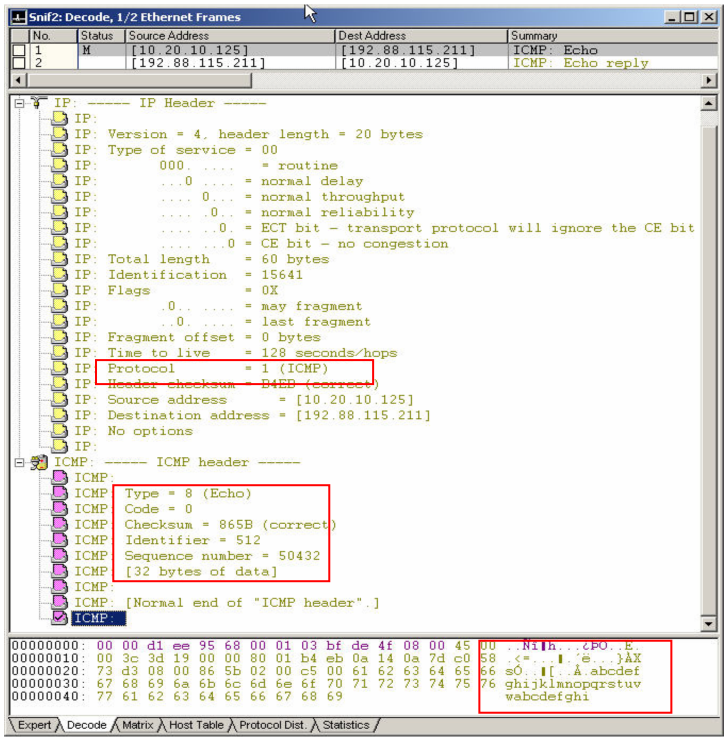
Notice the following:

The IP Protocol field = 1 for ICMP

The ICMP Type field = 8 for Echo
The ICMP Code = 0 (no code exists)

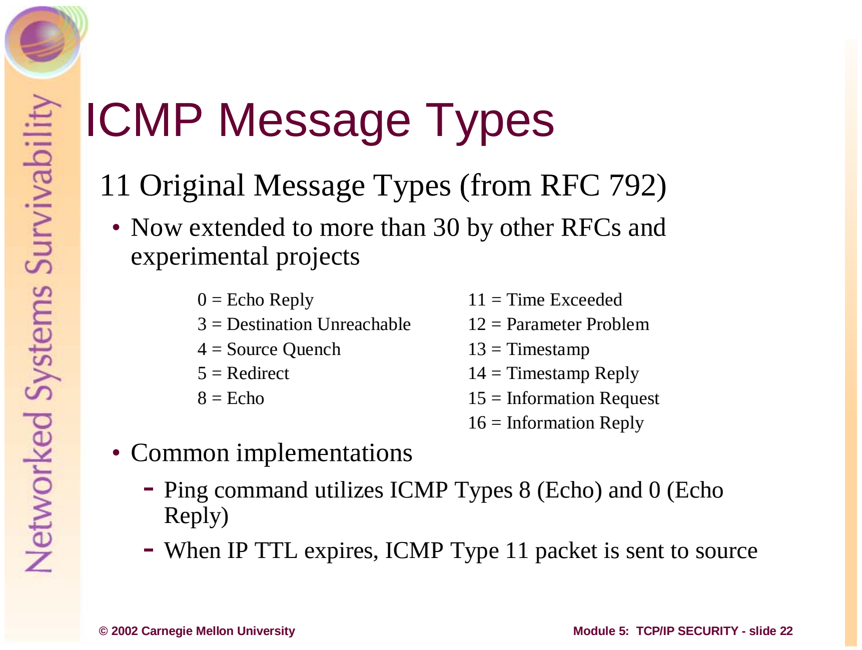
The Sequence number is a unique identifier for each Echo and Echo Reply session. In this case 50432 corresponds to this echo request packet and will be indentical in the subsequent Echo Reply packet only.

For data, the Ping program simply sends enough ASCII characters (a-w) to fill the standard packet size.


Snif2: Decode, 1/2 Ethernet Frames

No.	Status	Source Address	Dest Address	Summary
1	M	[10.20.10.125]	[192.88.115.211]	ICMP: Echo
2		[192.88.115.211]	[10.20.10.125]	ICMP: Echo reply

```
IP: ----- IP Header -----
  IP:
  IP: Version = 4, header length = 20 bytes
  IP: Type of service = 00
  IP:       000. ....   = routine
  IP:       ...0 .... = normal delay
  IP:       .... 0... = normal throughput
  IP:       .... .0.. = normal reliability
  IP:       .... ..0. = ECT bit - transport protocol will ignore the CE bit
  IP:       .... ...0 = CE bit - no congestion
  IP: Total length     = 60 bytes
  IP: Identification   = 15641
  IP: Flags            = 0X
  IP:       .0.. .... = may fragment
  IP:       ..0. .... = last fragment
  IP: Fragment offset = 0 bytes
  IP: Time to live     = 128 seconds/hops
  IP: Protocol         = 1 (ICMP)
  IP: Header checksum = D4ED (correct)
  IP: Source address      = [10.20.10.125]
  IP: Destination address = [192.88.115.211]
  IP: No options
  IP:
ICMP: ----- ICMP header -----
  ICMP:
  ICMP: Type = 8 (Echo)
  ICMP: Code = 0
  ICMP: Checksum = 865B (correct)
  ICMP: Identifier = 512
  ICMP: Sequence number = 50432
  ICMP: [32 bytes of data]
  ICMP:
  ICMP: [Normal end of "ICMP header".]
  ICMP:

00000000: 00 00 d1 ee 95 68 00 01 03 bf de 4f 08 00 45 00 ..Ñî▮h...¿ÞO..E.
00000010: 00 3c 3d 19 00 00 80 01 b4 eb 0a 14 0a 7d c0 58 .<=...▮.´ë...}ÀX
00000020: 73 d3 08 00 86 5b 02 00 c5 00 61 62 63 64 65 66 sÓ..▮[..Å.abcdef
00000030: 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 ghijklmnopqrstuv
00000040: 77 61 62 63 64 65 66 67 68 69                   wabcdefghi
```



ICMP Message Types

11 Original Message Types (from RFC 792)

- Now extended to more than 30 by other RFCs and experimental projects

0 = Echo Reply	11 = Time Exceeded
3 = Destination Unreachable	12 = Parameter Problem
4 = Source Quench	13 = Timestamp
5 = Redirect	14 = Timestamp Reply
8 = Echo	15 = Information Request
	16 = Information Reply

- Common implementations
  - Ping command utilizes ICMP Types 8 (Echo) and 0 (Echo Reply)
  - When IP TTL expires, ICMP Type 11 packet is sent to source

© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 22

Common ICMP Message Types and their descriptions:

ICMP Echo Reply (Ping): The most frequently used ICMP message is popularly implemented in a program called *ping*. It provides feedback to the sender on the state of IP connectivity and is often used as a debugging tool. Ping makes use of the ICMP ECHO REQUEST and ECHO REPLY parameters.

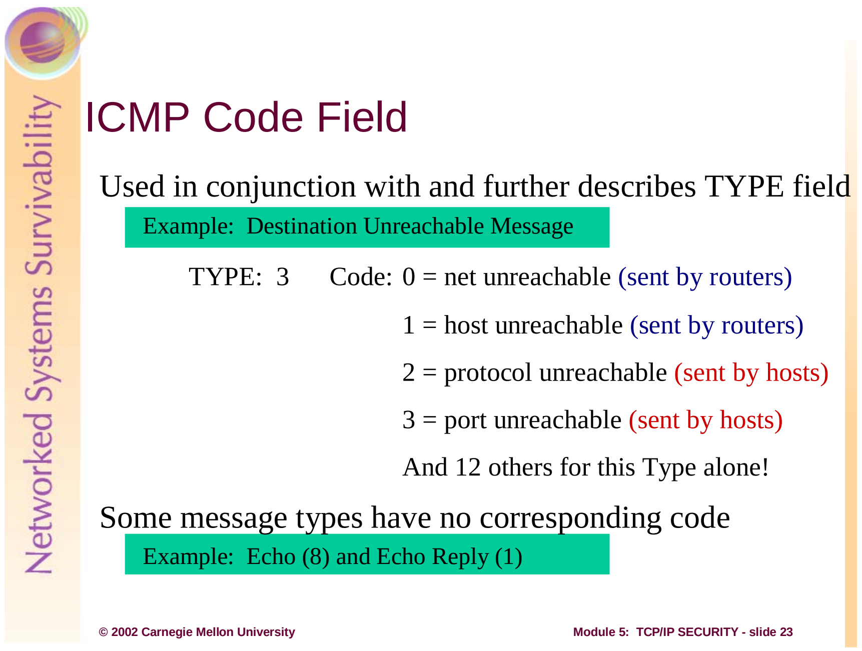
ICMP Redirects: The ICMP redirect message is sent by a gateway to the host, and instructs the host to use a different route when the router detects that its route is not as optimal as that of another router on the same network segment. If the gateway detects a better route for the IP datagram, it will send the host a redirect message with the address of the preferred gateway. TCP/IP will then send all traffic to this new IP address for another subnet.

ICMP Source Quench: IP provides a very basic form of flow control with the ICMP source quench message. The source quench message informs the originating host that the gateway or receiving host is being overrun and can't keep up with the traffic. The originating host then lowers the rate at which it sends datagrams to the receiving host, until it stops receiving "source quench" messages. After some time, the originating host may then gradually increase the rate at which it sends out datagrams.

ICMP Time Exceeded: Time Exceeded error messages are used to indicate that a forwarding or reassembly operation took too long to complete and that the reporting device is discarding the data. In order to provide more-detailed reporting, the Time Exceeded message provides two different Codes:

Time-to-Live Exceeded in Transit: An IP Packet's TTL has expired

Fragment Reassembly Time Exceeded: The IP's timer for reassembling fragments expired


Networked Systems Survivability

ICMP Code Field

Used in conjunction with and further describes TYPE field

Example: Destination Unreachable Message

TYPE: 3 Code: 0 = net unreachable (sent by routers)

1 = host unreachable (sent by routers)

2 = protocol unreachable (sent by hosts)

3 = port unreachable (sent by hosts)

And 12 others for this Type alone!

Some message types have no corresponding code

Example: Echo (8) and Echo Reply (1)

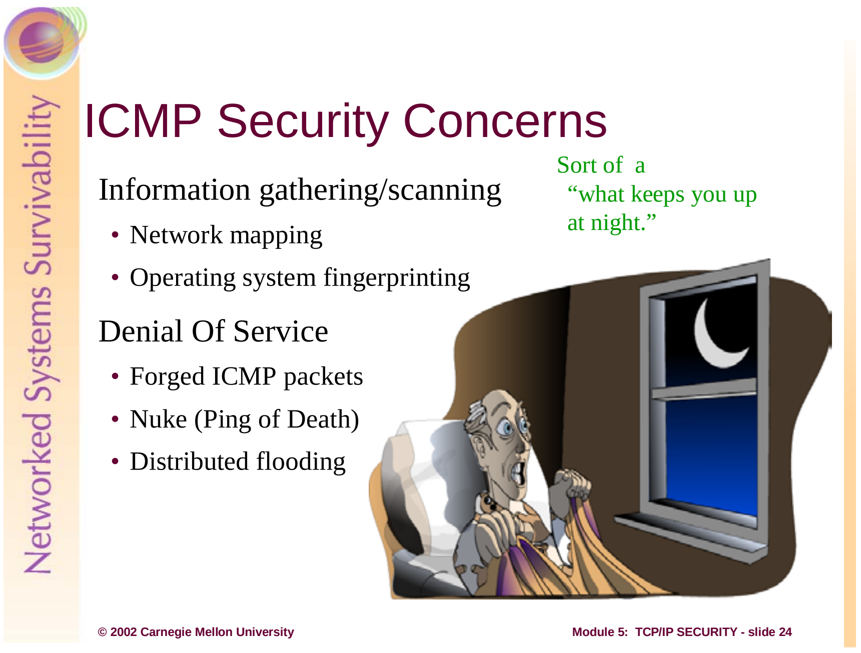
© 2002 Carnegie Mellon University — Module 5: TCP/IP SECURITY - slide 23

ICMP messages your computer can send or receive.

ICMP Messages with general message type, specific message code, and whether the message is a query or a response to a network event (error report).

Type	**Code**	**Description**	**Query/Error**
0	0	Echo reply	Q
3		**Destination Unreachable:**	
	0	Network Unreachable	E
	1	Host Unreachable	E
	2	Protocol Unreachable	E
	3	Port Unreachable	E
	4	Fragmentation needed and DF (flag) set	E
	5	Source Route Failed	E
	6	Destination Network Unknown	E
	7	Destination Host Unknown	E
	8	Source Host Isolated (obsolete)	E
	9	Destination Network Prohibited	E
	10	Destination Host Prohibited	E

Type	Code	Description	Query/Error
	11	Network Unreachable for TOS	E
	12	Host Unreachable for TOS	E
	13	Prohibited by Filtering	E
	14	Host Precedence Violation	E
	15	Precedence Cutoff in Effect	E
4	0	Source Quench	E
5		**Redirect:**	
	0	Redirect for Network	E
	1	Redirect for Host	E
	2	Redirect for TOS and Network	E
	3	Redirect for TOS and Host	E
8	0	Echo Request	Q
9	0	Router Advertisement	Q
10	0	Router Solicitation	Q
11		**Time Exceeded:**	
	0	TTL Exceeds 0 During Transit	E
	1	TTL Exceeds 0 During Reassembly	E
12		**Parameter Problem:**	
	0	IP Header Bad	E
	1	Required Option Missing	E
13	0	Timestamp Request	Q
14	0	Timestamp Reply	Q
15	0	Information Request (obsolete)	Q
16	0	Information Request Reply (obsolete)	Q
17	0	Address Mask Request	Q
18	0	Address Mask Reply	Q



Networked Systems Survivability

ICMP Security Concerns

Information gathering/scanning

- Network mapping
- Operating system fingerprinting

Denial Of Service

- Forged ICMP packets
- Nuke (Ping of Death)
- Distributed flooding

Sort of a "what keeps you up at night."

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 24

ICMP is widely used for Information Gathering and Scanning. It is commonplace to allow many different ICMP Type messages into and out of private networks. One of the most common is simple Echo and Echo Reply as implemented by the Ping program. There are multitudes of automated "ping sweep" type applications available on the Internet that map networks based on a specified range of IP addresses. These scanning techniques can potentially reveal the range of IP addresses within internal networks, what access control lists are being applied on routers and firewalls, and what the MTUs are on internal networks.

ICMP can also be used to determine which operating system is running—this is call OS Fingerprinting. One technique described by researcher Ofir Arkin[1] shows that Microsoft Windows hosts can be fingerprinted by sending invalid code fields inside of ICMP Timestamp Request packets. The manner in which the Windows systems respond to these requests are unique when compared to other operating systems. Arkin describes other similar techniques for fingerprinting Linux, Solaris and several other Unix-based systems.

ICMP packets do not include any authentication method for the recipient of the message. A clever hacker can forge ICMP packets and cause havoc in an unprepared network. The two greatest threats from malicious ICMP packets are denial-of-service attacks and impersonation or man-in-the-middle attacks. A forged destination-unreachable packet can isolate a computer from necessary services. Echo Request has been used by hackers to crash computers with a naive implementation of the ICMP protocols. Once a computer that performs an important service (such as a DNS server, file server, or web server) is out of the way, an ICMP redirect packet can point unwitting victims to the hacker's computer, where the hacker can accept authentication information (usernames, passwords, etc.).

Denial of service attacks primarily use either the ICMP "Time exceeded" or "Destination unreachable" messages. The "Time exceeded" message indicates that the Time-To-Live field in the IP header has expired; this can normally be caused by routing loops or trying to reach a host that is extremely distant. "Destination unreachable" messages can have several meanings (based on the ICMP Code field), but all basically indicate that packets cannot successfully be sent to the desired host. Both of these ICMP messages can cause a host to immediately drop a connection (this is the desired result if the ICMP

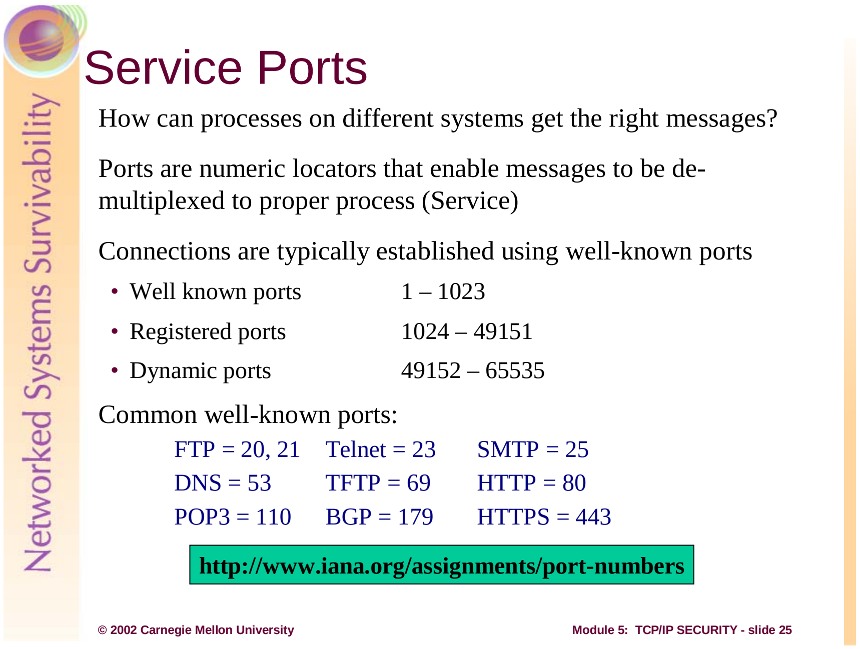
[1] http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf

message is legitimate). An attacker can make use of this by simply forging one of these ICMP messages, and sending it to one or both of the communicating hosts. Their connection will then be broken.

As mentioned above, ICMP messages can be used to intercept packets. The ICMP "Redirect" message is commonly used by routers when a host has mistakenly assumed the destination is not on the local network (and is thus attempting to send data allowed by the IP specification. The oversize packet is then sent to an unsuspecting systemthe packet via the gateway to the intended destination). If an attacker forges an ICMP "Redirect" message, it can cause another host to send packets for certain connections through the attacker's host.

The Ping of Death (sometimes referred to as Nuke attacks) uses a ping system utility to create an IP packet that exceeds the maximum 65,536 bytes of. This attack is obfuscated to a degree, because it requires that the packet be fragmented many times—remember the MTU for Ethernet is 1500 bytes. The results are varied but many systems may crash, hang, or reboot when they receive such a maliciously crafted packet[2].

[2] http://www.insecure.org/sploits/ping-o-death.html


Networked Systems Survivability

Service Ports

How can processes on different systems get the right messages?

Ports are numeric locators that enable messages to be de-multiplexed to proper process (Service)

Connections are typically established using well-known ports

• Well known ports	1 – 1023
• Registered ports	1024 – 49151
• Dynamic ports	49152 – 65535

Common well-known ports:

FTP = 20, 21	Telnet = 23	SMTP = 25
DNS = 53	TFTP = 69	HTTP = 80
POP3 = 110	BGP = 179	HTTPS = 443

http://www.iana.org/assignments/port-numbers

© 2002 Carnegie Mellon University   Module 5: TCP/IP SECURITY - slide 25

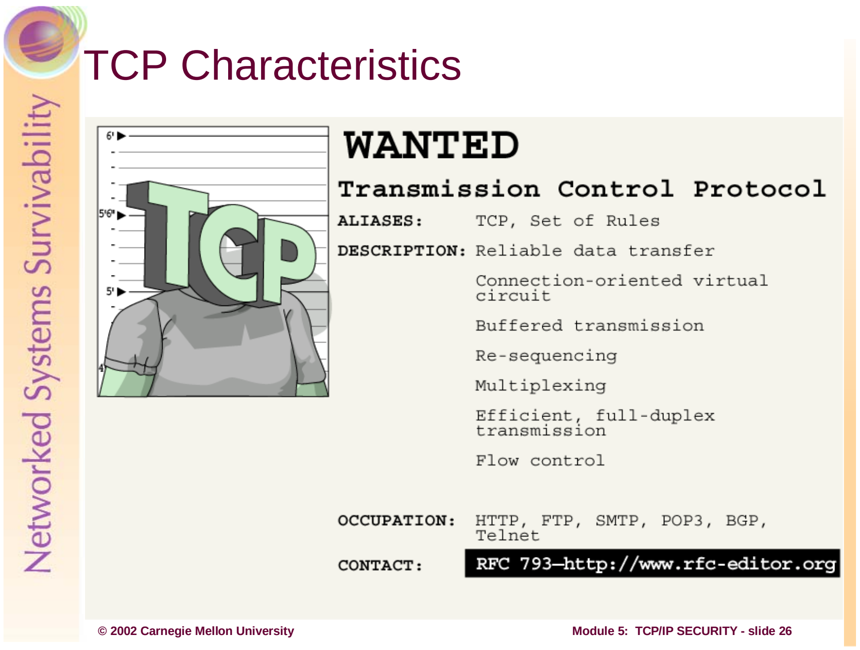
Higher-layer applications are referred to by a port identifier in TCP/UDP messages. The port identifier and IP address together form a *socket*, and the end-to-end communication between two hosts is uniquely identified on the Internet by the concatenating the source port, source address, destination port, destination address.

A 16-bit binary number specifies individual Port numbers. Port numbers in the range 0-1023 are called *Well Known Ports*. These port numbers are assigned to the server side of an application and, on most systems, can only be used by processes with a high level of privilege (such as root or administrator). Port numbers in the range 1024-49151 are called *Registered Ports*, and these are numbers that have been publicly defined as a convenience for the Internet community to avoid vendor conflicts. Server or client applications can use the port numbers in this range. The remaining port numbers, in the range 49152-65535, are called *Dynamic and/or Private Ports* and can be used freely by any client or server.

Some well-known port numbers include:

Port #	**Common Protocol**	**Service**	**Port #**	**Common Protocol**	**Service**
7	TCP	echo	80	TCP	http
9	TCP	discard	110	TCP	pop3
13	TCP	daytime	111	TCP	sunrpc
19	TCP	chargen	119	TCP	nntp
20	TCP	ftp-control	123	UDP	ntp
21	TCP	ftp-data	137	UDP	netbios-ns
23	TCP	telnet	138	UDP	netbios-dgm
25	TCP	smtp	139	TCP	netbios-ssn
37	UDP	time	143	TCP	imap
43	TCP	whois	161	UDP	snmp
53	TCP/UDP	dns	162	UDP	snmp-trap
67	UDP	bootps	179	TCP	bgp
68	UDP	bootpc	443	TCP	https (http/ssl)
69	UDP	tftp	520	UDP	rip
70	TCP	gopher	1080	TCP	socks
79	TCP	finger	33434	UDP	traceroute

A complete list of port numbers that have been assigned can be found in IANA's list of Port Numbers: http://www.iana.org/assignments/port-numbers. An implementation-specific list of supported port numbers and services can be found in the `services` file, generally found in the `/etc` (Linux/Unix), `c:\windows` (Windows 9x, ME), or `c:\winnt\system32\drivers\etc` (Windows NT, 2000) directory.


Networked Systems Survivability
6'
5'6"
5'
4'
TCP

TCP Characteristics

WANTED

Transmission Control Protocol

ALIASES:	TCP, Set of Rules
DESCRIPTION:	Reliable data transfer
	Connection-oriented virtual circuit
	Buffered transmission
	Re-sequencing
	Multiplexing
	Efficient, full-duplex transmission
	Flow control
OCCUPATION:	HTTP, FTP, SMTP, POP3, BGP, Telnet
CONTACT:	RFC 793—http://www.rfc-editor.org

© 2002 Carnegie Mellon University Module 5: TCP/IP SECURITY - slide 26

TCP is a connection-oriented, end-to-end reliable protocol. TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes an unreliable packet service from the lower level protocols—like IP which TCP utilizes. In principle, TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks. The continuous data stream flowing down from various applications (requesting network communications) is broken up into chunks called segments.

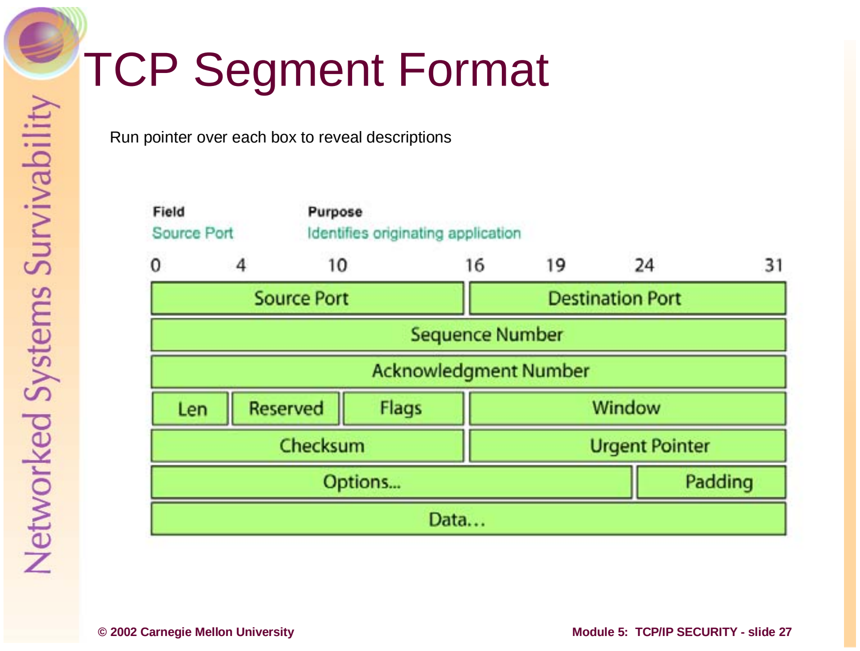
TCP is able to transfer a continuous stream of octets (bytes) in each direction between its users by packaging some number of octets into segments for transmission through the network. In general, TCP decides when to block and forward data through a rather elegant strategy called sliding windows.

TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by IP. TCP overcomes this by assigning a sequence number to each octet transmitted, and requiring a positive acknowledgment (ACK) from the other end. If the ACK is not received within a timeout interval, the data is retransmitted. On the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.

TCP provides a means for the receiver to govern the amount of data sent by the sender (flow control). It utilizes the sliding windows technique by returning a "window size" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.

To allow for many processes within a single Host to use TCP communication facilities simultaneously, TCP uses assigned port numbers. This port is combined with the network address from IP to form a socket. A pair of sockets uniquely identifies each connection. That is, a socket may be simultaneously used in multiple connections. The reliability and flow control mechanisms described above require that TCP initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection.

When two applications wish to communicate, their TCP's must first establish a connection (initialize the status information on each side). When their communication is complete, the connection is terminated or closed to free the resources for other uses. Since connections must be established between unreliable hosts and over the unreliable internet protocol, a handshake mechanism with sequence numbers is used to avoid erroneous initialization of connections.



TCP Segment Format

Networked Systems Survivability

Run pointer over each box to reveal descriptions

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 27

The TCP header fields:

Source Port and **Destination Port***:* Identify the source and destination ports to identify the end-to-end connection and higher-layer application.

Sequence Number*:* Contains the sequence number of this segment's first data byte in the overall connection byte stream; since the sequence number refers to a byte count rather than a segment count, sequence numbers in contiguous TCP segments are not numbered sequentially.

Acknowledgment Number*:* Used by the sender to acknowledge receipt of data; this field indicates the sequence number of the next byte expected from the receiver.

Data Offset: Points to the first data byte in this segment; this field, then, indicates the segment header length.

Control Flags*:* A set of flags that control certain aspects of the TCP virtual connection. The flags include:

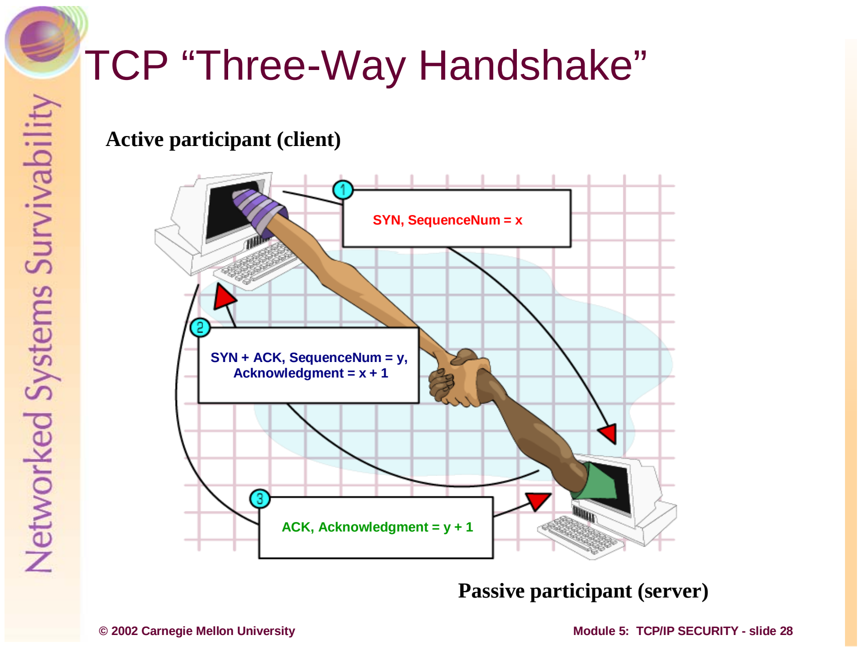
- **Urgent Pointer Field Significant (URG):** When set, indicates that the current segment contains urgent (or high-priority) data and that the Urgent Pointer field value is valid.
- **Acknowledgment Field Significant (ACK):** When set, indicates that the value contained in the Acknowledgment Number field is valid. This bit is usually set, except during the first message during connection establishment.
- **Push Function (PSH):** Used when the transmitting application wants to force TCP to immediately transmit the data that is currently buffered without waiting for the buffer to fill; useful for transmitting small units of data.
- **Reset Connection (RST):** When set, immediately terminates the end-to-end TCP connection.
- **Synchronize Sequence Numbers (SYN):** Set in the initial segments used to establish a connection, indicating that the segments carry the initial sequence number.
- **Finish (FIN):** Set to request normal termination of the TCP connection in the direction this segment is traveling; completely closing the connection requires one FIN segment in each direction.

Window: Used for flow control, contains the value of the *receive window size* which is the number of transmitted bytes that the sender of this segment is willing to accept from the receiver.

Checksum: Provides rudimentary bit error detection for the segment (including the header and data).

Urgent Pointer: Urgent data is information that has been marked as high-priority by a higher layer application; this data, in turn, usually bypasses normal TCP buffering and is placed in a segment between the header and "normal" data. The Urgent Pointer, valid when the URG flag is set, indicates the position of the first octet of nonexpedited data in the segment.

Options: Used at connection establishment to negotiate a variety of options; maximum segment size (MSS) is the most commonly used option and, if absent, defaults to an MSS of 536. Another option is Selective Acknowledgement (SACK), which allows out-of-sequence segments to be accepted by a receiver. The IANA maintains a list of all TCP Option Numbers: http://www.iana.org/assignments/tcp-parameters.



TCP “Three-Way Handshake”

Networked Systems Survivability
Active participant (client)
1
SYN, SequenceNum = x
2
SYN + ACK, SequenceNum = y, Acknowledgment = x + 1
3
ACK, Acknowledgment = y + 1
Passive participant (server)
© 2002 Carnegie Mellon University
Module 5: TCP/IP SECURITY - slide 28

For a connection to be established or initialized, the two TCPs (each host) must synchronize on each other's initial sequence numbers. This is done in an exchange of connection establishing segments carrying a flag bit called "SYN" (for synchronize) and the initial sequence numbers. As shorthand, segments carrying the SYN bit are also called "SYNs". Hence, the solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISN's.

The synchronization requires each side to send its own initial sequence number and to receive a confirmation of it in acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment.

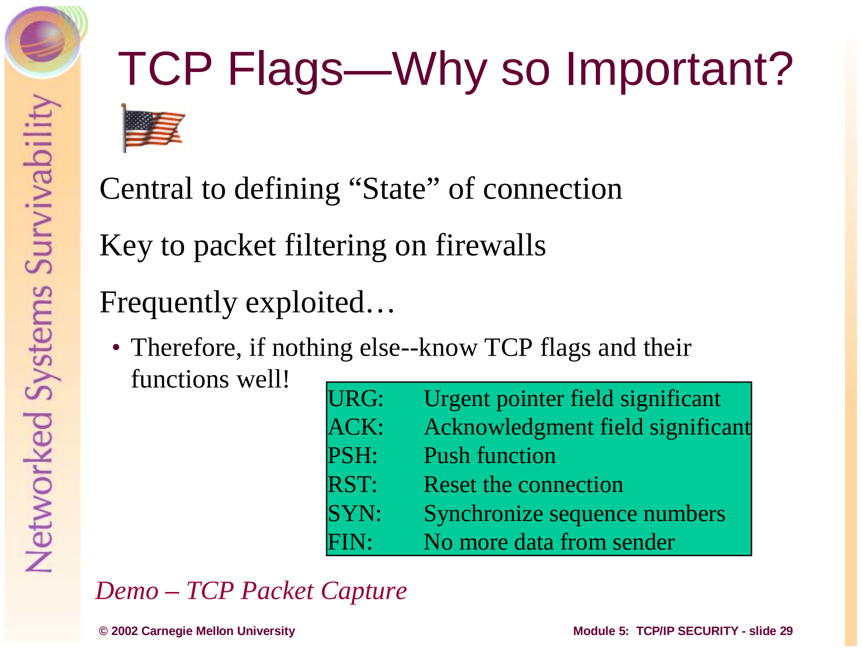
```
1) A --> B  SYN my sequence number is X
2) A <-- B  ACK your sequence number is X
3) A <-- B  SYN my sequence number is Y
4) A --> B  ACK your sequence number is Y
```

The simplest three-way handshake is shown in figure 1 below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Comments appear in parentheses. TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

```
    TCP A                                                  TCP B

1.  CLOSED                                                 LISTEN

2.  SYN-SENT    --> <SEQ=100><CTL=SYN>               --> SYN-RECEIVED

3.  ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED

4.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>       --> ESTABLISHED

5.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> --> ESTABLISHED

        Basic 3-Way Handshake for Connection Synchronization

                              Figure 1.
```

In line 2 of figure 1, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN, which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!) [RFC 793]



TCP Flags—Why so Important?

Central to defining "State" of connection

Key to packet filtering on firewalls

Frequently exploited…

- Therefore, if nothing else--know TCP flags and their functions well!

URG:	Urgent pointer field significant
ACK:	Acknowledgment field significant
PSH:	Push function
RST:	Reset the connection
SYN:	Synchronize sequence numbers
FIN:	No more data from sender

Demo – TCP Packet Capture

© 2002 Carnegie Mellon University Module 5: TCP/IP SECURITY - slide 29

Understanding what TCP flags are and how they are implemented is important to securing your networked information. As introduced on the previous slide, flags are used in part for identifying the current state of a TCP connection between 2 hosts. Remember, TCP connections involve synchronizing the connection (SYN), transmitting the data, and then finishing, or closing the connection (FIN). As requests are made between systems, acknowledgements (ACK) are transmitted back and forth to assure that both sides of the connection agree on the state of the connection.

Most static, and particularly, dynamic packet filtering implementations (applying access rules to individual packets—which is usually done on firewalls) utilize TCP flags to enforce access control policies. Because of this, it is important for administrators to understand how firewalls will filter packets based on which flags are set. This information will be covered in greater detail later in the course during the Firewalls module.

There exists, as part of the TCP specification, a level of interdependence between TCP Flags. An example of this is when a client sends an initial SYN segment. The client expects to receive a SYN/ACK segment back from the server and because it doesn't want to have to wait forever for this, it starts a timer. This is both good and bad. It's good because the client will not wait indefinitely for the connection state to change, however this can be exploited—typically with flooding the server with connection requests (see SYN floods next slide). There are other similar instances in the lifecycle of various TCP connection states that have no timer established. This can be a problem because that particular state can remain waiting for transition indefinitely.

As introduced above, TCP connections go through numerous state changes. These states are listed and briefly described below and are illustrated in the inserted diagrams. A complete explanation of each state is beyond the scope of this course, however it is highly recommended that security professionals understand this—refer to RFC 793.

The following is excerpted from RFC 793: A connection progresses through a series of states during its lifetime. The states are: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. Briefly the meanings of the states are:

LISTEN - represents waiting for a connection request from any remote TCP and port.

SYN-SENT - represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED - represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED - represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1 - represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2 - represents waiting for a connection termination request from the remote TCP.

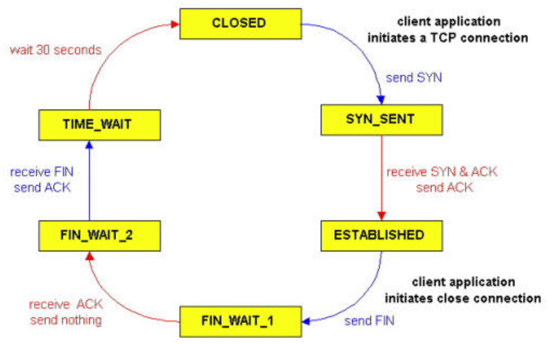
CLOSE-WAIT - represents waiting for a connection termination request from the local user.

CLOSING - represents waiting for a connection termination request acknowledgment from the remote TCP.

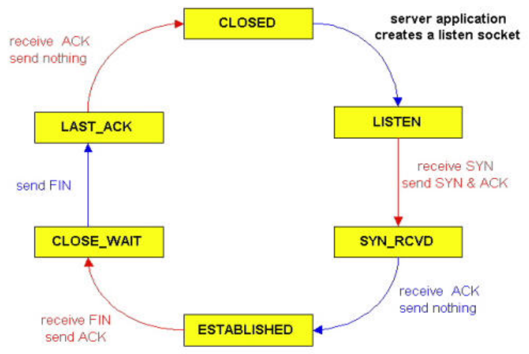
LAST-ACK - represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

TIME-WAIT - represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request.

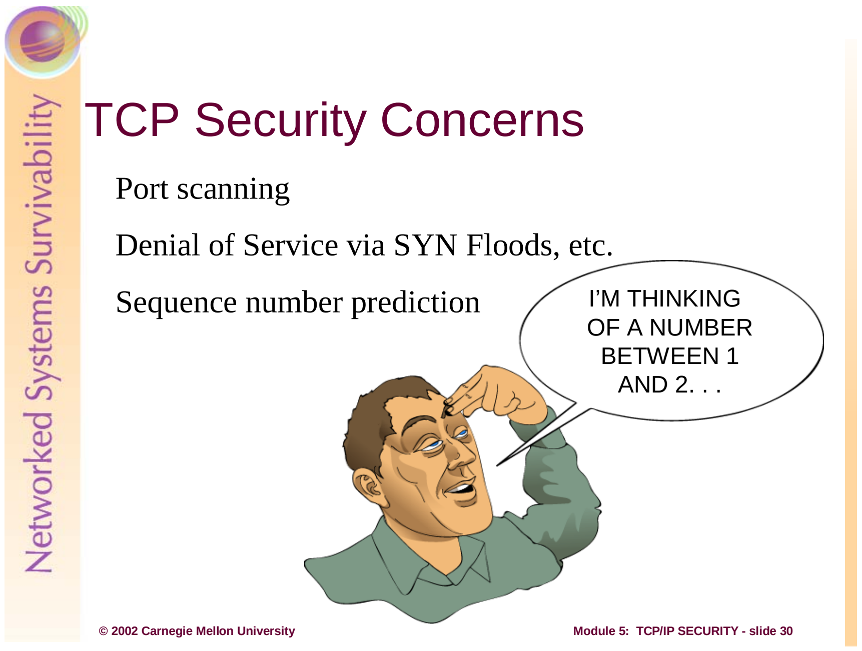
CLOSED - represents no connection state at all.



TCP Client Connection Lifecycle



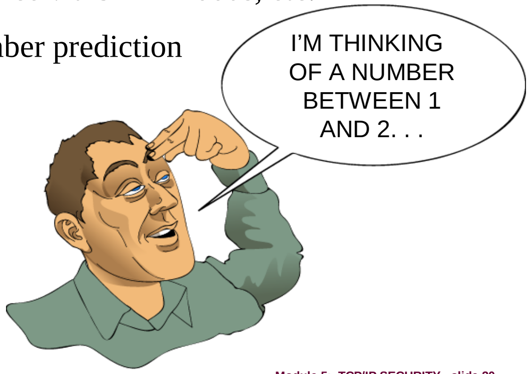
TCP Server Connection Lifecycle



Networked Systems Survivability

TCP Security Concerns

- Port scanning
- Denial of Service via SYN Floods, etc.
- Sequence number prediction



© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 30

Port Scanning is one of the most popular reconnaissance techniques attackers use to discover services they can break into. All machines connected to the Internet may be running services that listen at well-known and not so well-known ports. By port scanning the attacker finds which ports are available (i.e., being listened to by a service). Essentially, a port scan consists of sending a message to each port, one at a time. The kind of response received indicates whether the port is used and can therefore be probed further for weakness. Two TCP scanning methods are described below:

SYN scanning: This technique is also called *half-open* scanning, because a TCP connection is not completed. A SYN packet is sent (as if we are going to open a connection), and the target host responds with a SYN+ACK, this indicates the port is listening, and an RST indicates a non- listener. The server process is never informed by the TCP layer because the connection did not complete—that is to say, the client didn't send the final ACK to the server to complete the three-way handshake.

FIN scanning: The typical TCP scan attempts to open connections (at least part way). Another technique sends erroneous packets at a port, expecting that open listening ports will send back different error messages than closed ports. The scanner sends a FIN packet, which should close a connection that is open. Closed ports reply to a FIN packet with a RST. Open ports, on the other hand, ignore the packet in question. This is required TCP behavior. If no service is listening at the target port, the operating system will generate an error message. If a service is listening, the operating system will silently drop the incoming packet. Therefore, silence indicates the presence of a service at the port.

The same TCP behavior described in SYN scanning is exploited in another (very popular!) attack called SYN flooding. The potential for abuse arises at the point where the server system has sent an acknowledgment (SYN-ACK) back to client but has not yet received the ACK message. The server has built in its system memory a data structure (a buffer or queue) describing all pending connections.

This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open connections. Creating half-open connections is easily accomplished with IP spoofing. The attacking system sends SYN messages to the victim server system; these appear to be legitimate but in fact reference a client system that is unable to respond to the SYN-ACK messages. This means that the final ACK message will never be sent to the victim server system. The half-open connections data structure on the victim server system will eventually fill; then the system will be unable to accept any new incoming connections until the table is emptied out. Normally there is a timeout associated with a pending connection, so the half-open connections will eventually expire and the victim server system will recover. However, the attacking system can simply continue sending IP-spoofed packets requesting new connections faster than the victim system can expire the pending connections. In most cases, the victim of such an attack will have difficulty in accepting any new incoming network connection. In some cases, the system may exhaust memory, crash, or be rendered otherwise inoperative. The location of the attacking system is obscured because the source addresses in the SYN packets are often implausible. When the packet arrives at the victim server system, there is virtually no way to determine its true source. Similar attacks are employed with the FIN flag set—which makes static packet filtering notably ineffective. The latest abuses of this vulnerability are distributed flooding attacks—where many hacked systems act as the source of the flooding.

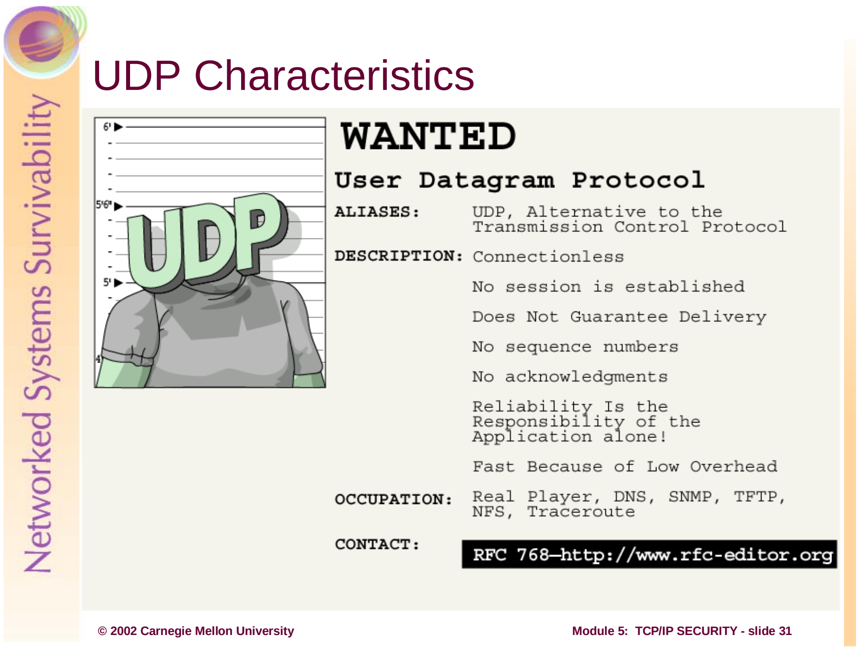
TCP sequence number prediction is a well-known vulnerability that was first described in 1985.

The TCP sequence number is a 32-bit counter. In order to distinguish between different connections between the same sender and receiver, it is important that the sequence numbers do not start at 0 or any other fixed number each time a connection is opened. Hence, it is important that the first byte of data from the sender to the receiver is using a random sequence number. Many current implementations increment the sequence number by a finite amount every second.

More precisely, RFC 793 specifies that the 32-bit counter be incremented by 1 in the low-order position about every 4 microseconds. Instead, Berkeley-derived Unix kernels increment it by a constant every second, and by another constant for each new connection. Thus, if you open a connection to a machine, you know to a very high degree of confidence what sequence number it will use for its next connection. And therein lies the attack [RFC 1948]. Many vendors have changed the algorithms employed in their TCP/IP implementations to defend against this, however recent research has called into question whether or not these mitigation strategies are effective.[3] Studies have been done that analyze the susceptibility of specific operating systems to sequence number prediction.[4] Another concern is how effectively faulty TCP/IP stacks are patched. For Windows 95 and 98, Microsoft utilized the TCP/IP stack implementation directly from the open source code of BSD 4.4. This code was patched by the original BSD developers, however, Microsoft had to develop their own patches for their stack—which may or may not have been a priority—because of the closed (proprietary) nature of the software.

[3] http://www.kb.cert.org/vuls/id/498440

[4] http://razor.bindview.com/publish/papers/tcpseq.html



UDP Characteristics

Networked Systems Survivability
6'
5'6"
5'
UDP

WANTED

User Datagram Protocol

ALIASES:	UDP, Alternative to the Transmission Control Protocol
DESCRIPTION:	Connectionless
	No session is established
	Does Not Guarantee Delivery
	No sequence numbers
	No acknowledgments
	Reliability Is the Responsibility of the Application alone!
	Fast Because of Low Overhead
OCCUPATION:	Real Player, DNS, SNMP, TFTP, NFS, Traceroute
CONTACT:	RFC 768–http://www.rfc-editor.org

© 2002 Carnegie Mellon University Module 5: TCP/IP SECURITY - slide 31

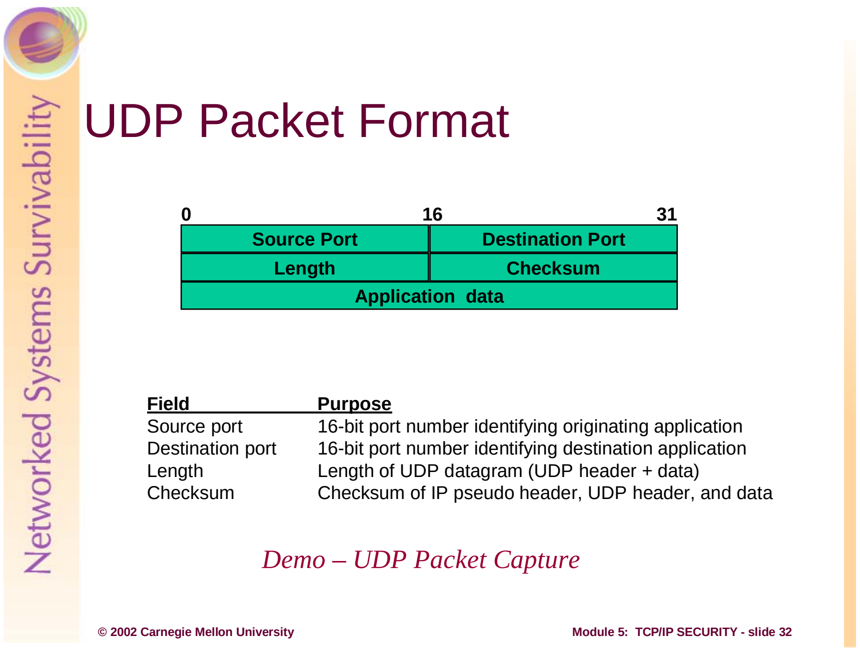
UDP is one of the two main protocols to reside on top of IP. It offers service to the user's network applications. Example network applications that use UDP are: Network File System (NFS) and Simple Network Management Protocol (SNMP). The service is little more than an interface to IP.

UDP is a connectionless packet delivery service that does not guarantee delivery. UDP does not maintain an end-to-end connection with the remote UDP module; it merely pushes the packet out on the net and accepts incoming packets off the net. UDP adds two values to what is provided by IP. One is the multiplexing of information between applications based on port number. The other is a checksum to check the integrity of the data and header information.

The path of communication between an application and UDP is through UDP ports. These ports are numbered, beginning with zero. An application that is offering service (the server) waits for messages to come in on a specific port dedicated to that service. The server waits patiently for any client to request service. For instance, the TFTP server, always waits (listens) on port 69. There can be only one TFTP service per computer (unless a TFTP server has been configured to use a non-standard port number) because there is only one UDP port number 69. This port number is well known; it is a fixed number, an internet assigned number. If a TFTP client wants service, it sends its request to UDP port number 69 on the destination computer.

When an application sends data out through UDP it arrives at the far end as a single unit. For example, if an application does 5 writes to the UDP port, the application at the far end will do 5 reads from the UDP port. Also, the size of each write matches the size of each read. UDP preserves the message boundary defined by the application. It never joins two application messages together, or divides a single application message into parts.

An incoming IP packet with an IP header type field indicating "UDP" is passed up to the UDP module by IP. When the UDP module receives the UDP packet from IP it examines the UDP checksum. If the checksum field value is zero (16 off bits), it means that checksum was not calculated by the sender and can be ignored. UDP checksumming was designed to be optional for performance reasons. If Ethernet, which has a robust error checking method, is the only underlying network between two communicating systems; then you may have UDP option out of performing checksums. It is however, recommended that UDP checksum generation always be enabled because changes in routing tables can send packets over less reliable media access protocol implementations. If UDP performs a checksum at both ends and the packet contains no errors, the destination port number is examined and if an application is bound to that port, the message is queued for the application to read. If the checksum is performed and errors are detected, the packet is discarded. If no checksum is performed, than UDP simply queues the data (which may have errors in it) for the application. Additionally, if the incoming UDP packets arrive faster than the application can read them, thereby overflowing the queue's maximum value, UDP packets are discarded. UDP will continue to discard packets until there is space in the queue.



UDP Packet Format

Field	Purpose
Source port	16-bit port number identifying originating application
Destination port	16-bit port number identifying destination application
Length	Length of UDP datagram (UDP header + data)
Checksum	Checksum of IP pseudo header, UDP header, and data

Demo – UDP Packet Capture

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 32

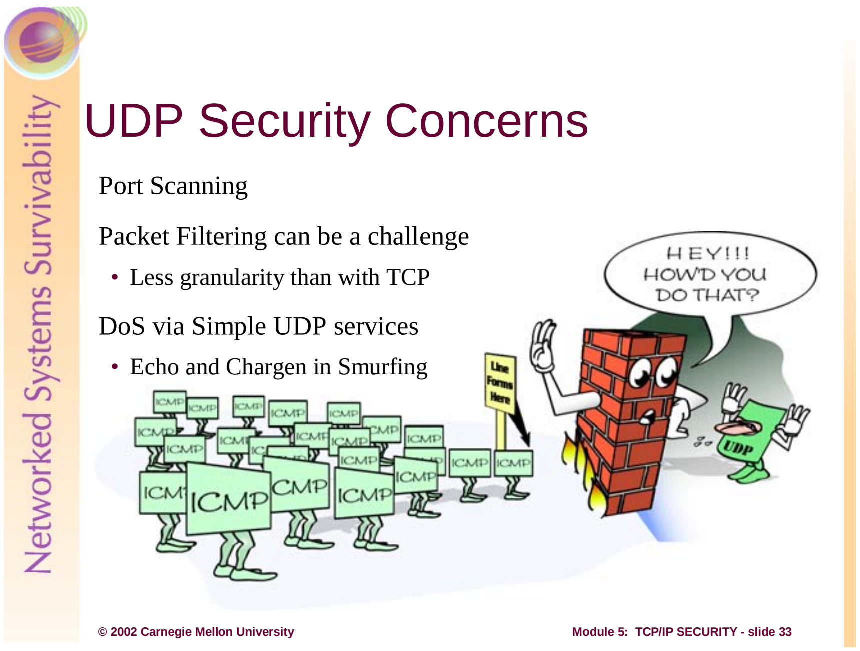
The fields of a UDP packet are:

Source Port: Identifies the UDP port being used by the sender of the packet; use of this field is optional in UDP and may be set to 0.

Destination Port: Identifies the port used by the packet receiver.

Length: Indicates the total length of the UDP packet including data.

Checksum: Provides rudimentary bit error detection for the packet (including the header and data)



Networked Systems Survivability

UDP Security Concerns

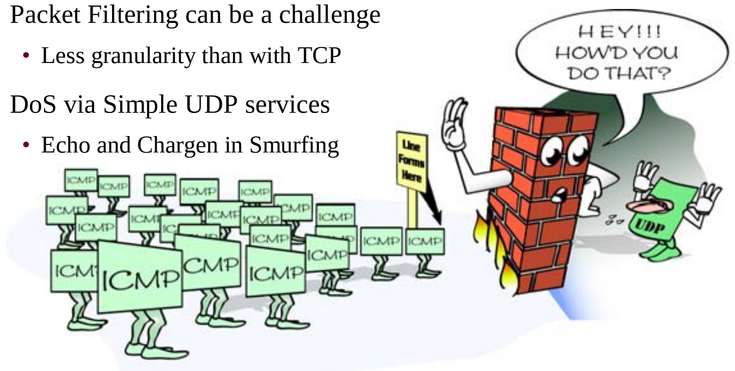
Port Scanning

Packet Filtering can be a challenge

- Less granularity than with TCP

DoS via Simple UDP services

- Echo and Chargen in Smurfing



© 2002 Carnegie Mellon University

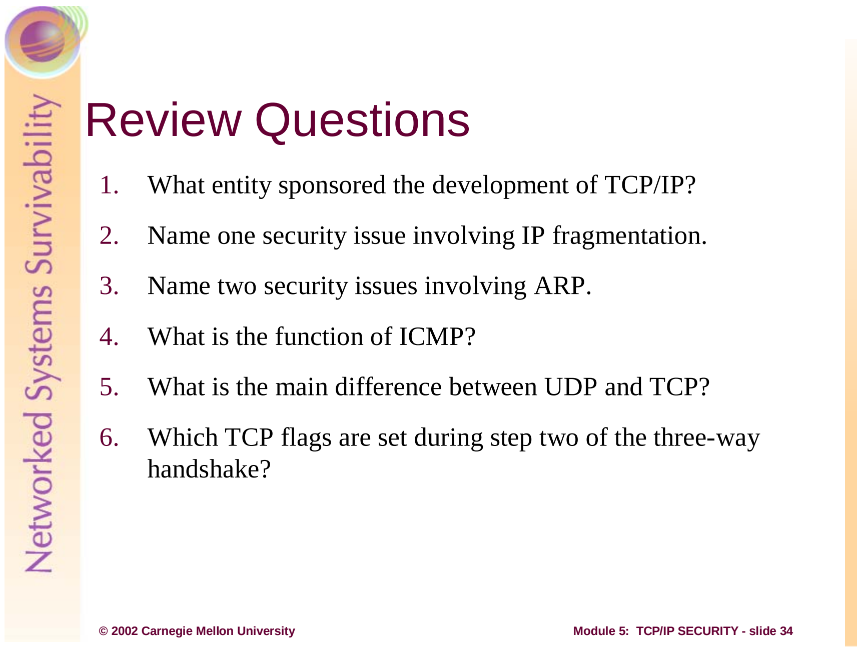
Module 5: TCP/IP SECURITY - slide 33

Port scanning usually means scanning for TCP ports, which are connection-oriented and therefore give good feedback to the attacker. UDP responds in a different manner. In order to find UDP ports, the attacker generally sends empty UDP datagrams. If the port is listening, the service should send back an error message or ignore the incoming datagram. If the port is closed, then most operating systems send back an "ICMP Port Unreachable" message. Thus, you can find out if a port is NOT open, and by exclusion determine which ports are open. Neither UDP packets, nor the ICMP errors are guaranteed to arrive, so UDP scanners of this sort must also implement retransmission of packets that appear to be lost (or you will get a bunch of false positives). Also, this scanning technique is slow because of compensation for machines that implement the suggestions of RFC 1812 and limit ICMP error message rate. For example, the Linux kernel limits destination unreachable message generation to 80 per 4 seconds, with a 1/4 second penalty if that is exceeded.

Some people think UDP scanning is pointless -- not so. Consider the Solaris rpcbind hole (Sun Microsystems Security Bulletin Bulletin Number: #00167, April 8, 1998). Rpcbind can be found hiding on an undocumented UDP port somewhere above 32770. So it doesn't matter that port 111 (SUN Remote Procedure Call) is blocked by the firewall. But can you find which of the more than 30,000 high ports it is listening on? With a UDP scanner you can! There are other applications that operate similarly.

Packet filtering UDP packets on firewalls is not as eloquent or complex as with TCP. Because UDP is connectionless and therefore devoid of any “stateful” information, packet filters are forced to be very black and white with how they enforce access control policies. Non-Stateful firewalls can basically deny or permit UDP traffic based on port number alone—that is the extent of granularity. As a result, some attackers favor using UDP with either randomly generated dynamic port numbers, or static ports between 49152 and 65535.

Some UDP ports and their assigned services are particularly vulnerable to DOS attacks. By sending a crafted UDP packet to a host that is listening on UDP port 19 (chargen) with a source port of UDP 7 (echo) you’ll start a flood of network traffic that will grow exponentially. Chargen or Character Generator is a testing service that generates and sends all ASCII characters to the requester. UDP Echo (not to be confused with ICMP Echo) simply sends all of the data it receives back to the requester. The number of connections increases as each service tries to complete its purpose—hence, the floodgates are opened to another form of DOS flooding.

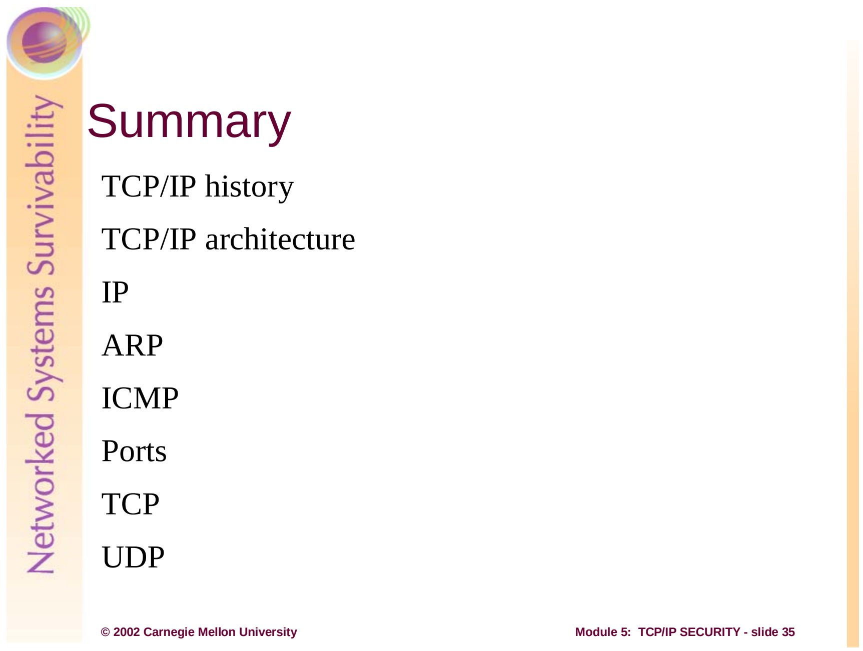

Networked Systems Survivability

Review Questions

1. What entity sponsored the development of TCP/IP?
2. Name one security issue involving IP fragmentation.
3. Name two security issues involving ARP.
4. What is the function of ICMP?
5. What is the main difference between UDP and TCP?
6. Which TCP flags are set during step two of the three-way handshake?

© 2002 Carnegie Mellon University Module 5: TCP/IP SECURITY - slide 34

1. What entity sponsored the development of TCP/IP
   Answer: DARPA
2. Name 1 security issue involving IP fragmentation.
   Answer: Can confuse firewalls and Intrusion Detection Systems
3. Name 2 security issues involving ARP.
   Answer: Man-in-the-Middle attacks, ARP Spoofing and Flooding
4. What is the function of ICMP?
   Answer: To report errors in the communications environment of IP and to provide a means of performing network diagnostics and testing
5. What is the main difference between UDP and TCP?
   Answer: UDP is connectionless whereas TCP is connection-oriented
6. Which TCP flags are set during step 2 of the Three-way Handshake?
   Answer: SYN and ACK



Networked Systems Survivability

Summary

TCP/IP history

TCP/IP architecture

IP

ARP

ICMP

Ports

TCP

UDP

© 2002 Carnegie Mellon University

Module 5: TCP/IP SECURITY - slide 35

Bibliography:

[Cisco] *Understanding TCP/IP* Available at:
http://www.cisco.com/univercd/cc/td/doc/product/iaabu/centri4/user/scf4ap1.htm

[ISOC 2001] *A Brief History of the Internet* Available at:
http://www.isoc.org/internet/history/brief.shtml

RFC 768 UDP Available at: http://www.rfc-editor.org/rfc/rfc768.txt

RFC 791 IP Available at: http://www.rfc-editor.org/rfc/rfc791.txt

RFC 792 ICMP Available at: http://www.rfc-editor.org/rfc/rfc792.txt

RFC 793 TCP Available at: http://www.rfc-editor.org/rfc/rfc793.txt

RFC 826 ARP Available at: http://www.rfc-editor.org/rfc/rfc826.txt

[Used throughout] Comer, Douglas E. *Internetworking with TCP/IP, Vol. 1: Principles, Protocols, and Architecture.* 3rd edition. New York:" Prentice-Hall, 1995.

[Used throughout] *TCP/IP Illustrated, Vol. 1: The Protocols*. Richard Stevens Reading, MA: Addison-Wesley, 1994.

[Used throughout] "TCP/IP Security," Chris Chambers, Justin Dolske, and Jayaraman Iyer (Department of Computer and Information Science, Ohio State University):
http://www.linuxsecurity.com/resource_files/documentation/tcpip- security.html

[Used throughout] Security Problems in the TCP/IP Protocol Suite," Bellovin, S. M. (*Computer Communication Review*, April 1989): http://www.ja.net/CERT/Bellovin/TCP-IP_Security_Problems.html

[whatis 2000] *What is ARP? Available at:*
http://whatis.techtarget.com/definition/0,,sid9_gci213780,00.html