TCP/IP: sniffing, ARP attacks, IP fragmentation

Network Security

Lecture 3

Recap and overview

Last time

- TCP/IP
  - IP
  - Ethernet
  - ARP

Today

- Attacks
  - Sniffing
  - Spoofing
  - Hijacking (ARP)
- Tools/libraries
  - Libnet, libpcap
- TCP/IP
  - Fragmentation

Exercise

- Alice (192.168.1.1) wants to send an IP datagram to Bob (192.168.1.2)
- What happens? (fill in the blanks)

LAN attacks

Attack	Security violation	Attacker goal
Sniffing	Confidentiality	Access to information
Spoofing	Authenticity	Impersonation of trusted host
Hijacking	Confidentiality, Integrity, Authenticity	Impersonation, access to information
Denial of Service	Availability	Disruption

Network sniffing

- The attacker sets his/her network interface in *promiscuous* mode so that all packets can be received (not only those directed to the attacker's host)
- Can access all the traffic on the segment
- Note: sniffing on University network is a "disciplinary offence"

Network sniffing

- Many protocols (e.g., POP, TELNET, HTTP, IMAP) transfer sensitive information (e.g., authentication credentials) in the clear
- By sniffing the traffic, it is possible to collect credentials, files, content of visited web pages, emails, etc.
- Many tools available

tcpdump

- Tool to sniff and analyze the traffic on a network segment
- One of the “standard” network 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 set the interface in promiscuous mode (regular users can read traffic data saved in a file)

tcpdump: command line options

- -i: use the given 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 (0 means catch whole packets)
- -n: do not convert addresses to names
- -x: print the data of each packet in hex

tcpdump: filters

- If a filter expression is provided, tpcdump only processes packets matching the expression
- Expression consists of one or more primitives
- Primitives are composed of a qualifier and a value
- Operators can be used to create complex filter expressions

tcpdump filters – cont’d

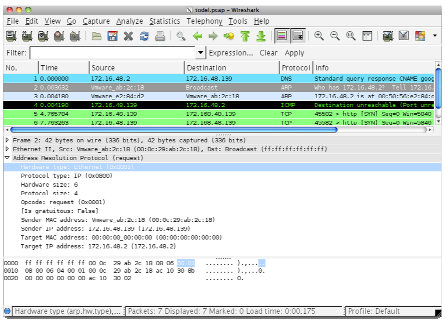
Qualifiers

- Type
  - host (host 192.168.0.1)
  - net (net 192.168)
  - port (port 80)
- Dir: direction of traffic
  - src (src host 192.168.0.1)
  - dst
- Proto: protocol of interest
  - Ether (ether src host 00:0c:29:ab:2c:18)
  - ip
  - arp

Operators

- Logical: and, or, not
  - src host 192.168.0.01 and dst host google.com
- Relational: <, >, >=, <=, =, !=
- Binary: +, -, *, /, &, |
- Data: proto[expr:size]
  - expr: offset
  - size: # bytes of interest
  - ip[0] & 0xf > 5: filters IP datagrams with options
  - arp[7] = 2: ARP replies

Wireshark



Detecting sniffers

- Sniffers work by putting the network interface in promiscuous mode
- Ifconfig (BSD Unix/Apple Output)
- ```
  $ ifconfig en1
  en1: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        ether d8:a2:5e:ab:cd:ef
        inet 10.4.59.191 netmask 0xffff0000 broadcast 10.4.255.255
        media: autoselect
        status: active
  ```
- On recent Linux versions, this will not (always) work due to changes in how the state of the interface is maintained in the kernel that have not been ported back to tools
  - Instead, read interface flags from /sys filesystem
  - If flags & 0x100 then interface is in promiscuous mode (/include/linux/if.h)

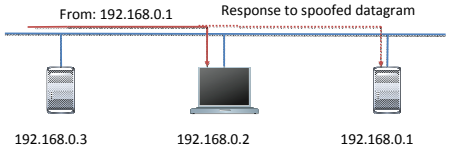
```
# cat /sys/class/net/eth0/flags
0x1003
# tcpdump –i eth0 &
# cat /sys/class/net/eth0/flags
0x1103
```
```

Detecting sniffers – cont'd

- Remote detection is difficult since sniffers are typically passive programs
- Suspicious DNS lookups
  - Sniffer attempts to resolve names associated with IP address (e.g., tcpdump without –n option)
  - Generate traffic to/from IP addresses and detect attempts to resolve their names
  - `$ ping 173.194.37.104`
    `16:27:38.657863 IP 172.16.48.130 > 173.194.37.104: ICMP echo request, id 21009, seq 1, length 64`
    `16:27:38.659014 IP 172.16.48.139.57105 > 172.16.48.2.53: 20764+ PTR? 104.37.194.173.in-addr.arpa. (45)`
- Latency
  - Since NIC is in promiscuous mode, it will need to process *every* packet
  - Analyze response time of host A (e.g., sending ping packets)
  - Generate lots of traffic to other hosts and analyze response time of host A

IP spoofing

- A host impersonates another host by sending a datagram that has the address of some other host as the source address
- The attacker sniffs the network looking for replies from the attacked host
- Replies would be directed to the spoofed host



IP spoofing goals

- Impersonate sources of security-critical information (e.g., a DNS server or an NFS server)
- Exploit address-based authentication
- Many tools available

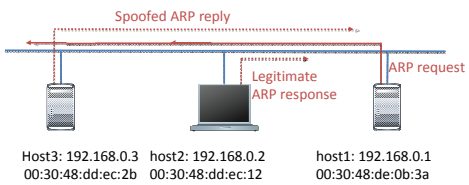
Hijacking

- Sniffing/Spoofing are the bases for hijacking
- The attacker waits for an client request
- Then, races against the legitimate host to produce a reply that will be accepted by the client
- ARP, UDP, and TCP-based variations of this attack

Hijacking ARP

- ARP does not provide any means of authentication
- Racing against the queried host it is possible to provide a fake IP address/link-level address mapping
- Fake ARP queries can be used to store wrong ARP mappings in a host cache, in certain configurations
  - In Linux, /proc/sys/net/ipv4/conf/*/arp_accept should be set to 1
- In both cases, the net effect is the redirection of traffic to the attacker
  - Denial of service (DoS)
  - Man-in-the-middle attack (MITM)

Hijacking ARP



Switched Ethernet

- Switched Ethernet does not allow direct sniffing
- ARP spoofing can be used to bypass this protection
- MAC flooding
  - Switches maintain a table with MAC address/port mappings
  - In some cases, flooding the switch with bogus MAC addresses will overflow table memory and revert the behavior from "switch" to "hub"
- MAC spoofing
  - Reconfigure the 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

Capturing and forging packets

Libpcap

- Library to sniff network traffic
- Allows to easily filter and process packets
- http://www.tcpdump.org/
- Good tutorial: http://www.tcpdump.org/pcap.html

libnet

- Library to forge packets
- Useful to send raw or malformed packets
- https://github.com/sam-github/libnet
- Good tutorial: http://repura.livejournal.com/31673.html
- Documentation:
- http://libnet.sourcearchive.com/documentation/1.1.2.1-4/

libpcap

- pcap_lookupdev
  - Finds a device to sniff from
- pcap_open_live
  - Opens a device (returns a handle)
- pcap_compile and pcap_setfilter
  - Compile a tcpdump-like traffic filter and applies it
- pcap_loop
  - Registers a callback to be invoked for every received packet

libpcap

- `void pcap_handler(u_char *user, const struct pcap_pkthdr *hdr, const u_char *pkt)`
- The pcap packet header (`hdr`) contains basic information about the packet
  - When it was captured (`ts`)
  - The length of the portion that was captured (`caplen`)
  - The length of the packet (`len`)
- The actual packet (`pkt`) is returned as a pointer to memory
- Packets can be parsed by "casting" it to appropriate protocol-specific structures
- Remember that endianness is important!
  - ntohs, ntohl
  - htons, htonl

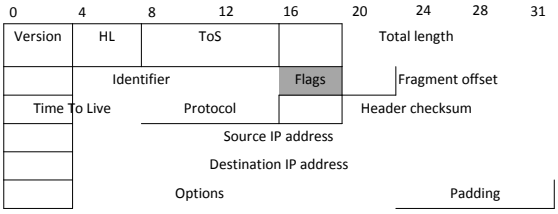
libnet

- libnet_init
  - Initializes the library
- libnet_autobuild_ethernet
  - Builds ethernet header
- libnet_autobuild_arp
- libnet_autobuild_ipv4
- libnet_build_tcp
- …
- libnet_write
  - Writes packet to wire
- Libnet_clear_packet
  - Clears current packet

IP fragmentation

- When a datagram is encapsulated in lower level protocols (e.g., Ethernet) it may be necessary to split the datagram in smaller portions
- Link layer specifies a Maximum Transmission Unit (MTU): the size in bytes of the largest data unit that can be transferred on the layer
- If datagram size is bigger than MTU, then fragmentation
- Fragmentation can be performed at source host or at an intermediate step in the datagram delivery
- If the datagram has the "don't fragment" flag set, an ICMP error message is sent back to the source host

IP fragmentation



Flags:
- bit 0: reserved
- bit 1: don't fragment (DF)
- bit 2: more fragments (MF)

IP fragmentation

- If datagram can be fragmented
  - Header is copied in each fragment
  - The MF flag is set in all fragments except the last one
  - The fragmentation offset field contains the position of the fragment with respect to the original datagram (as 8-byte units)
  - Total length field is adjusted to match the fragment size
- Each fragment is delivered as a separate datagram
- If one fragment is lost, entire datagram is discarded

IP fragmentation

```
$ ifconfig en1
 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
$ ping -c1 -s 1472 192.168.0.1
00:00:00.000000 IP (tos 0x0, ttl 64, id 43907, offset 0, flags [none], proto ICMP (1), length
    1500) 192.168.0.100 > 192.168.0.1: ICMP echo request, id 9497, seq 0, length 1480

$ ping -c1 -s 1473 192.168.0.1
00:00:45.969839 IP (tos 0x0, ttl 64, id 35311, offset 0, flags [+], proto ICMP (1), length
    1500) 192.168.0.100 > 192.168.0.1: ICMP echo request, id 20249, seq 0, length 1480
00:00:00.000708 IP (tos 0x0, ttl 64, id 35311, offset 1480, flags [none], proto ICMP (1),
    length 21) 192.168.0.100 > 192.168.0.1: icmp

$ ping -c1 -s 1473 -D 192.168.0.1
ping: sendto: Message too long

$ ping -c1 -s 1472 -D www.google.com
PING www.l.google.com (74.125.230.83): 1472 data bytes
36 bytes from adsl211-220.aknet.it (194.242.211.220): frag needed and DF set (MTU 1492)
00:00:18.349153 IP (tos 0x0, ttl 64, id 24038, offset 0, flags [DF], proto ICMP (1), length
    1500) 192.168.0.100 > 74.125.230.83: ICMP echo request, id 28185, seq 0, length 1480
00:00:00.056466 IP (tos 0xc0, ttl 63, id 24038, offset 0, flags [none], proto ICMP (1), length
    56) 194.242.211.220 > 192.168.0.100: ICMP 74.125.230.83 unreachable - need to frag (mtu
    1492), length 36
```


IP fragmentation attacks: ping of death

- The offset of the last fragment is such that the total size of the reassembled datagram is bigger than the maximum allowed size
- Static buffer in the kernel is overflowed, causing a kernel panic
- Circa 1998

The Linux 2.0.24 patch:

```
/*
 * Attempt to construct an
 * oversize packet.
 */
if(ntohs(iph->tot_len) +
   (int)offset > 65535)
{
  skb->sk = NULL;
  frag_kfree_skb(skb,
FREE_READ);
  ip_statistics.IpReasmFails++;
  return NULL;
}
```


IP fragmentation attacks: evasion

- Firewalls and intrusion detection systems analyze incoming datagrams using the information contained in both the datagram header and the datagram payload (TCP ports, UDP ports, SYN and ACK flags in the TCP header)
- An attacker may use fragmentation to avoid filtering
  - Some firewalls may make a decision on the first fragment and let the other fragments through (based on the datagram ID)
  - Payload data can be divided in multiple fragments
    - Setup flags can be postponed in successive fragments
    - Setup flags (SYN/ACK) can be overwritten by using overlapping fragments

IP fragmentation attacks: evasion

- An attacker may use fragmentation to avoid detection
  - Some intrusion detection systems (IDS) may not reassemble datagrams
  - An IDS may reassemble datagram differently than target system
- Tools exist to fragment traffic in different ways
  - http://monkey.org/~dugsong/fragroute/

IP indirect delivery (routing)

We have already seen direct delivery

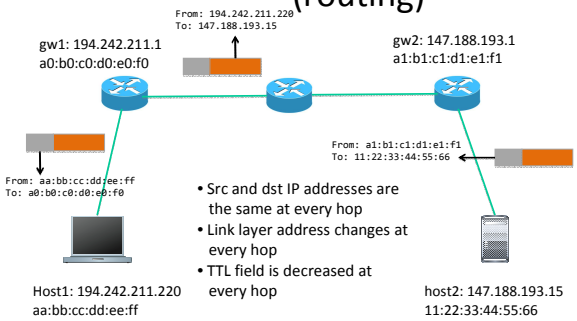
If two hosts are in different physical networks the IP datagram is encapsulated in a lower level protocol and delivered to the directly connected gateway

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

IP indirect delivery (routing)



- Src and dst IP addresses are the same at every hop
- Link layer address changes at every hop
- TTL field is decreased at every hop

Routing

Hop-by-hop routing

The delivery route is determined by the gateways that are traversed in the delivery process

Source routing

The sender (source of datagram) specifies a partial or complete list of gateways the datagram must pass through in sequence before being delivered to destination (IP option)

Hop-by-hop routing

The information needed to deliver datagram to next hop is stored in the routing table

```
$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt
Iface
172.16.48.0     0.0.0.0         255.255.255.0   U         0 0          0
eth0
0.0.0.0         172.16.48.2     0.0.0.0         UG        0 0          0
eth0
```

Flags

U: route is up
H: target is host
G: use gateway
D: dynamically installed by daemon or redirect message
M: modified by daemon or redirect message

Hop-by-hop routing

Search for a matching host address

Search for a matching network address

Search for a default entry

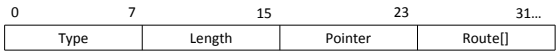
If a match is not found a message of "host unreachable" or "network unreachable" is returned (by the kernel or by a remote gateway by using ICMP)

Routing tables can be set

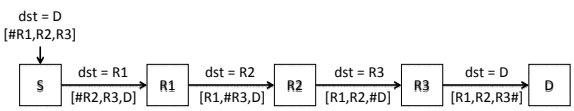
Statically, at boot or by using `route` command

Dynamically, using routing protocols

Source routing



- **Type**:
  - 131 Loose Source and Record Route (LSRR)
  - 137 Strict Source and Record Route (SSRR)
- **Length**: total length of the option
- **Pointer**: pointer into the route data (4, 8, etc.)
- **Route** data: array of IP addresses



Source routing

- Frequently blocked by routers

```
$ traceroute www.google.co.uk
traceroute to www.google.co.uk
(173.194.37.104), 30 hops max, 40
byte packets
 1  rita-rw (147.188.193.6)  1.455
ms  1.401 ms  1.372 ms
...
16  lhr14s02-in-f104.1e100.net
(173.194.37.104)  9.097 ms  9.556 ms
9.522 ms

$ traceroute -g 147.188.193.6
www.google.co.uk
 1  * * *
...
30  * * *
```

- Perfect for spoofing attacks
  - alice: 1.1.1.1
  - bob: 2.2.2.2
  - malice: 6.6.6.6
- Malice sends a datagram with alice's spoofed source address (1.1.1.1) to bob (2.2.2.2) and specifies malice's gateway (6.6.6.1) in the source routing list
- When bob responds, its data passes through malice's gateway