TCP/IP: libnet + libpcap, fragmentation, IP routing

Network Security

Lecture 4

Announcement

- Homework 1 is due at noon!
- Homework 2 is out!
- Demonstrator: Ian Batten
  Office hour: Thursdays 2pm, Room 245
- Before next lecture, read S. Bellovin,
  Security Problems in the TCP/IP Protocol Suite

Recap and overview

Last time

- Sniffing
  - Tcpdump
  - wireshark
- Spoofing
- Hijacking
  - ARP

Today

- Libraries to sniff and forge packets
- IP fragmentation
- IP routing
  - Source routing
  - Hop-by-hop routing
- ICMP
  - Protocol
  - Attacks

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

- Build packets from upper layer to lower layer (e.g., first ARP, then Ethernet header))
- Write packet on the network
- libnet_build_* vs. libnet_autobuild_*
  - More control on each field
- One packet is build in each opened libnet context. If need to send more packets:
  - libnet_clear_packet: clears current packet and starts from scratch
  - Use tags to update parts of packet that were modified

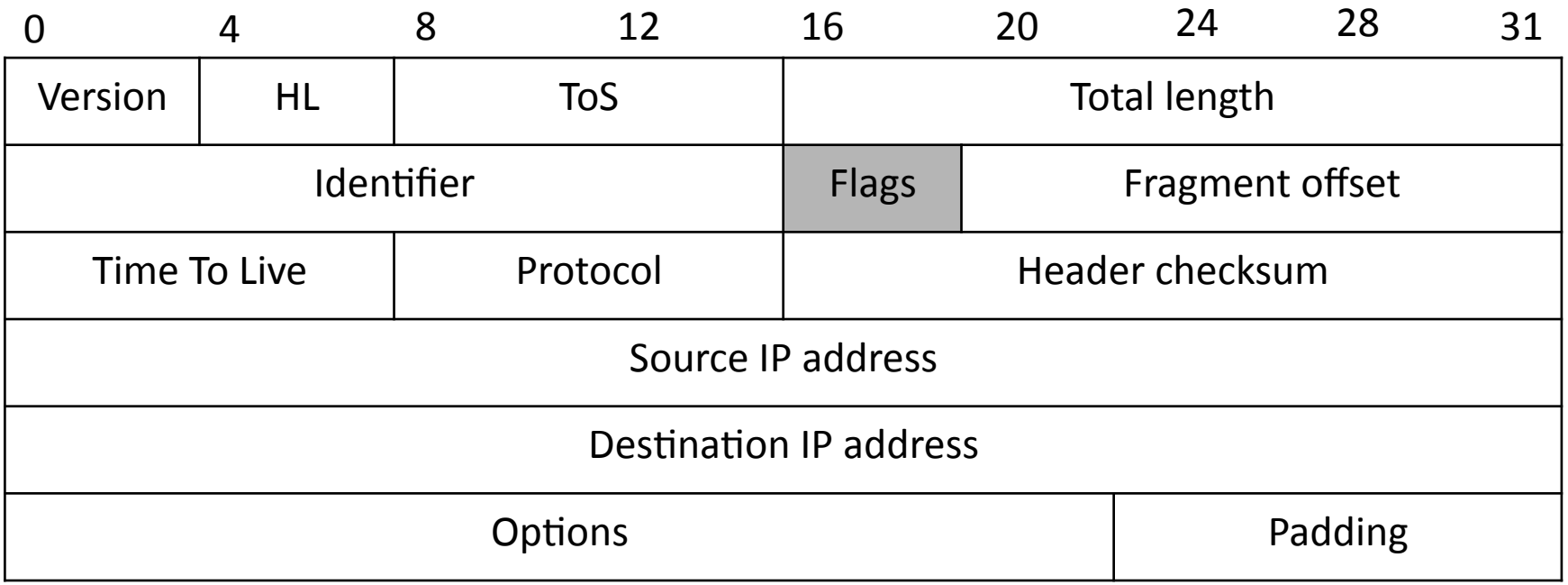
Sample code

- http://www.cs.bham.ac.uk/~covam/teaching/2010/netsec/lecture-04.tar.gz
  - Example of using libnet
  - Example of using libpcap
  - Makefile to compile source files

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
- Reassembly is done only at the destination host
- 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
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500

$ 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

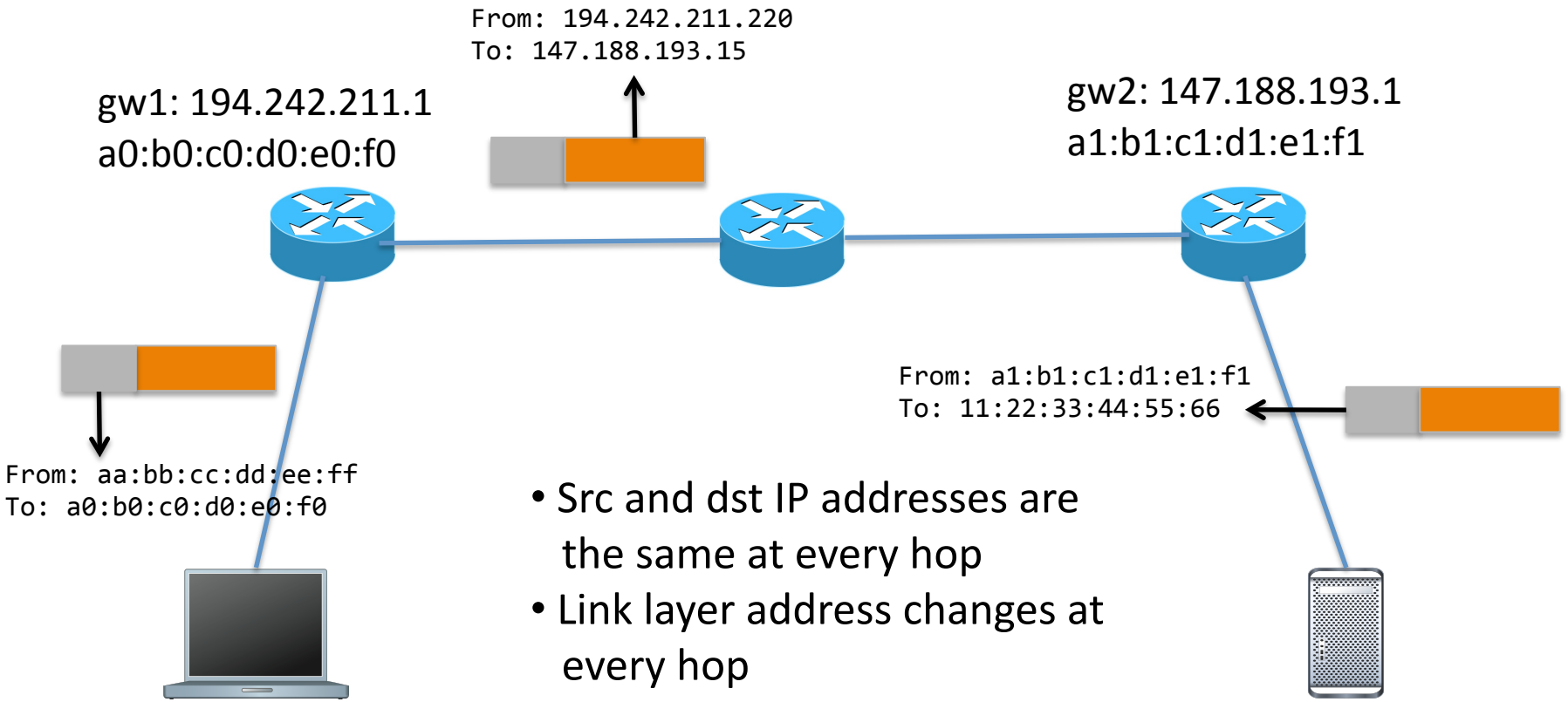
IP fragmentation attacks: evasion

- 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

Host1: 194.242.211.220
aa:bb:cc:dd:ee:ff

host2: 147.188.193.15
11:22:33:44:55:66

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

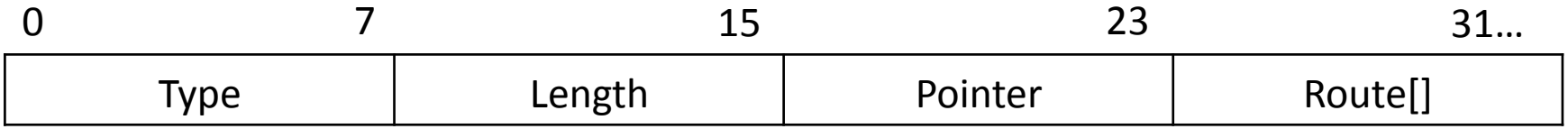
```
$ netstat –rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt
Iface
172.16.48.0     0.0.0.0         255.255.255.0   U         0 0          0
eth0
0.0.0.0         172.16.48.2     0.0.0.0         UG        0 0          0
eth0
```

- 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

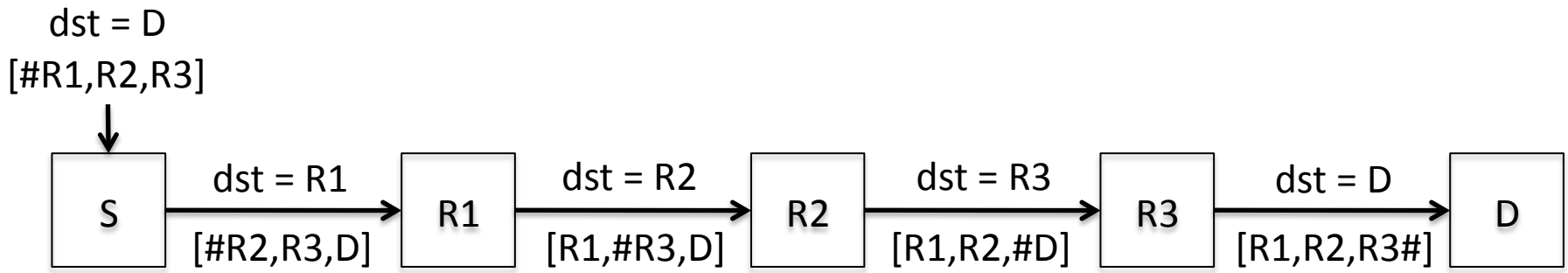
Hop-by-hop routing

- 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

NEXT ON

Take away points

- libnet + libpcap
- IP fragmentation
- Routing
  - Source routing
  - Hop-by-hop routing

Next time

- ICMP-based attacks
- UDP