CSE 40567 / 60567: Computer Security

Network Security 2
Homework #6 has been released. It is due 4/4 at 11:59PM

See Assignments Page on the course website for details
Network Eavesdropping
Good places for eavesdropping

At the router (capture traffic from multiple networks)

Personal computers

Multi-user servers

Unprotected Wireless APs
tcpdump

• Trusty Unix packet sniffer (command line interface)
• Requires root privilege to capture network traffic on an interface

List interfaces on which tcpdump can listen:

```
# tcpdump -D
1.eth0
2.eth1
3.usbmon1 (USB bus number 1)
4.usbmon2 (USB bus number 2)
5.usbmon3 (USB bus number 3)
6.usbmon4 (USB bus number 4)
7.usbmon5 (USB bus number 5)
8.usbmon6 (USB bus number 6)
9.usbmon7 (USB bus number 7)
10.usbmon8 (USB bus number 8)
11.any (Pseudo-device that captures on all interfaces)
12.lo
```
tcpdump

Listen on interface eth0 (first ethernet device)

# tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
16:25:12.840690 IP dhcp-140-247-178-194.fas.harvard.edu. 5900 > c-73-168-122-160.hsd1.in.comcast.net.61227: Flags [P.], seq 13216:13680, ack 1, win 272, options [nop,nop,TS val 4196170389 ecr 1186915574], length 464...

...
Targeted sniffing is often more useful.

Capture packets to a particular destination:
\[
\text{tcpdump} \ -n \ dst \ host \ 192.168.1.1
\]

Capture packets from a particular source:
\[
\text{tcpdump} \ -n \ src \ host \ 192.168.1.1
\]

Capture packets to / from a particular host:
\[
\text{tcpdump} \ -n \ host \ 192.168.1.1
\]

Capture packets to / from a particular network:
\[
\text{tcpdump} \ -n \ net \ 192.168.1.0/24
\]
tcpdump

Capture packets related to a specific port:
   tcpdump -n port 22

Capture packets related to a range of tcp ports:
   tcpdump -n tcp portrange 1-1023

Capture packets to a range of udp ports:
   tcpdump -n udp portrange 1-1023

Capture ARP packets:
   tcpdump -v arp

Capture ICMP packets:
   tcpdump -v icmp
DNS resolution

$ host nd.edu

# tcpdump -n udp port 53

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
15:29:01.393162 IP 140.247.178.194.38502 > 128.103.1.7.53: 39751+ A? nd.edu. (24)
15:29:01.394643 IP 140.247.233.163.53 > 140.247.178.194.38502: 39751 1/0/0 A 52.6.129.12 (40)
15:29:01.396338 IP 140.247.233.163.53 > 140.247.178.194.7260: 13012 0/1/0 (68)
15:29:01.396791 IP 140.247.178.194.43203 > 140.247.233.163.53: 29425+ MX? nd.edu. (24)
15:29:01.397814 IP 140.247.233.163.53 > 140.247.178.194.43203: 29425 2/0/0 MX mail-mx3-prod-v.cc.nd.edu. 50, MX mail-mx4-prod-v.cc.nd.edu. 50 (91)
nc

Swiss army knife of socket tools

$ nc 192.168.0.1 80 ← raw connection to web server

- Outbound or inbound connections, TCP or UDP, to or from any ports
- Full DNS forward/reverse checking, with appropriate warnings
- Ability to use any local source port
- Ability to use any locally configured network source address
- Built-in port-scanning capabilities, with randomization
- Built-in loose source-routing capability
- Slow-send mode, one line every N seconds
- Hex dump of transmitted and received data
- Tunneling mode which permits user-defined tunneling
Web session

$ nc www.google.com 80
HEAD / HTTP/1.0

HTTP/1.0 200 OK
Date: Sun, 14 Feb 2016 21:11:52 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See https://www.google.com/support/accounts/answer/151657?hl=en for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie:
NID=76=ci6oYJcjmFMJr8zqYUKr88yr6B9isPOAcmKEk5k2NgLm7IOAcO6pw9iJxbb8w9MsET2p-J-i0V0b2VBdvnTU8H6XZI1qW6dT9ZxwcXw9-TbEvcyTYQLRUIesU6_YAd8Ualo3fanw; expires=Mon, 15-Aug-2016 21:11:52 GMT; path=/; domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Web session (3-way handshake)

# tcpdump -n tcp port 80

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes

16:04:55.589296 IP 140.247.178.194.37612 > 4.53.56.118.80: Flags [S], seq 108294392, win 14600, options [mss 1460,sackOK,TS val 4217466076 ecr 0,nop,wscale 7], length 0

16:04:55.590276 IP 4.53.56.118.80 > 140.247.178.194.37612: Flags [S.], seq 3684023987, ack 108294393, win 28960, options [mss 1460,sackOK,TS val 741439636 ecr 4217466076,nop,wscale 7], length 0

16:04:55.590298 IP 140.247.178.194.37612 > 4.53.56.118.80: Flags [.], ack 1, win 115, options [nop,nop,TS val 4217466076 ecr 741439636], length 0
Web session (data transmission)

16:05:00.023831 IP 140.247.178.194.37612 > 4.53.56.118.80: Flags [P.], seq 1:17, ack 1, win 115, options [nop,nop,TS val 4217467185 ecr 741439636], length 16

16:05:00.024328 IP 4.53.56.118.80 > 140.247.178.194.37612: Flags [], ack 17, win 227, options [nop,nop,TS val 741444070 ecr 4217467185], length 0

16:05:00.163811 IP 140.247.178.194.37612 > 4.53.56.118.80: Flags [P.], seq 17:18, ack 1, win 115, options [nop,nop,TS val 4217467220 ecr 741444070], length 1

16:05:00.164245 IP 4.53.56.118.80 > 140.247.178.194.37612: Flags [], ack 18, win 227, options [nop,nop,TS val 741444210 ecr 4217467220], length 0

16:05:00.234189 IP 4.53.56.118.80 > 140.247.178.194.37612: Flags [P.], seq 1:626, ack 18, win 227, options [nop,nop,TS val 741444280 ecr 4217467220], length 625

16:05:00.234195 IP 140.247.178.194.37612 > 4.53.56.118.80: Flags [], ack 626, win 124, options [nop,nop,TS val 4217467237 ecr 741444280], length 0
Web session (termination)

16:05:00.234356 IP 4.53.56.118.80 > 140.247.178.194.37612: Flags [F.], seq 626, ack 18, win 227, options [nop,nop,TS val 741444280 ecr 4217467220], length 0

16:05:00.234401 IP 140.247.178.194.37612 > 4.53.56.118.80: Flags [F.], seq 18, ack 627, win 124, options [nop,nop,TS val 4217467237 ecr 741444280], length 0

16:05:00.234958 IP 4.53.56.118.80 > 140.247.178.194.37612: Flags [., ack 19, win 227, options [nop,nop,TS val 741444281 ecr 4217467237], length 0
Security vulnerability: plaintext passwords

Without encryption, passwords are trivial to recover with sniffing.

Large problem for custom web apps and mobile applications
  ▪ Less of a concern for common protocols these days

Several protocols still in widespread use are susceptible to this: HTTP, POP3, SNMP, and FTP
Example: ftp

# tcpdump -X -n tcp port 21

16:21:33.236417 IP 140.247.178.194.50384 > 69.163.224.12.21: Flags [P.],
seq 1:14, ack 27, win 115, length 13
  0x0000: 4510 0035 87c0 4000 4006 4d89 8cf7 b2c2   E..5..@@.M.....
  0x0010: 45a3 e00c c4d0 0015 5af2 b588 99e8 a61b   E........Z.........
  0x0020: 5018 0073 6591 0000 5553 4552 2077 616c   P..se...USER.wal-
  0x0030: 7465 720d 0a ter..

16:21:35.252626 IP 140.247.178.194.50384 > 69.163.224.12.21: Flags [P.],
seq 14:27, ack 61, win 115, length 13
  0x0000: 4510 0035 87c2 4000 4006 4d87 8cf7 b2c2   E..5..@@.M.....
  0x0010: 45a3 e00c c4d0 0015 5af2 b595 99e8 a63d   E........Z........=
  0x0020: 5018 0073 6591 0000 5041 5353 2066 6f6f   P..se...PASS.foo-
  0x0030: 6261 720d 0a bar..
Security vulnerability: online behavior profiling

• Even if passwords are encrypted, you can learn a lot about a user’s behavior
  ‣ Blackmail
  ‣ Research for future social engineering attack against the user

• Learn about network topology
  ‣ Servers (internal / external)
  ‣ Clients (low-hanging fruit)
Wireshark

Image credit: https://www.wireshark.org

User-friendly front-end for packet sniffing
Wireshark
PCAP Interface

`libpcap` in Unix; `WinPcap` in Windows

C API, with wrapper support for many other languages:

- Python
- Ruby
- Rust
- Java
- C#
- Go

Powering many tools:

- `tcpdump`
- `NMAP.ORG`
- `SNORT`
libpcap (device handling)

/* Define the device */
dev = pcap_lookupdev(errbuf);
if (dev == NULL) {
    fprintf(stderr, "Couldn’t find default device: %s\n", errbuf);
    return(2);
}

/* Find the properties for the device */
if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
    fprintf(stderr, "Couldn’t get netmask for device %s: %s\n", dev, errbuf);
    net = 0;
    mask = 0;
}
libpcap (session setup)

/* Open the session in promiscuous mode */
handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL) {
    fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
    return(2);
}

/* Compile and apply the filter */
if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
    fprintf(stderr, "Couldn't parse filter %s: %s\n", filter_exp, pcap_geterr(handle));
    return(2);
}

if (pcap_setfilter(handle, &fp) == -1) {
    fprintf(stderr, "Couldn't install filter %s: %s\n", filter_exp, pcap_geterr(handle));
    return(2);
}
libpcap (capture and close)

/* Grab a packet */
packet = pcap_next(handle, &header);

/* Print its length */
printf("Captured a packet with length of [%d]\n", header.len);

/* And close the session */
pcap_close(handle);
return(0);
Wireless Eavesdropping

- Open access points
- WEP attacks
  - Less common these days, but occasionally WEP-enabled devices are encountered
- Known weaknesses in WPA and WPA2
  - Authenticated attacker may be able to sniff the network
Kismet (Unix)
https://www.kismetwireless.net/

- 802.11 sniffing
- Standard PCAP logging
- Client/Server modular architecture
- Plug-in architecture to expand core features
- Multiple capture source support
- Live export of packets to other tools via tun/tap virtual interfaces
- Distributed remote sniffing via light-weight remote capture
- XML output for integration with other tools
KisMac2 (OS X)

https://github.com/IGRSoft/KisMac2

Mac version of Kismet, with a friendlier UI
What is floating out on the ether?

Packets captured at Eddy St. Commons (IP changed to protect the innocent):


User listening to music
What is floating out on the ether?

Packets captured at Eddy St. Commons (IP changed to protect the innocent):

17:40:04.618312 CF +QoS IP 10.10.10.1.53045 >
dns1.nd.edu.domain: 39098+ A? app.snapchat.com. (34)

17:40:04.629288 CF +QoS IP 10.10.10.1.53045 >
dns1.nd.edu.domain: 39098+ A? app.snapchat.com. (34)

User doing some messaging
What is floating out on the ether?

Packets captured at Eddy St. Commons (IP changed to protect the innocent):

17:40:10.272639 CF +QoS IP 10.10.10.1.64141 > s3-1-w.amazonaws.com.http: Flags [S], seq 2601035886, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 768497473 ecr 0,sackOK,eol], length 0

17:40:10.272724 CF +QoS IP 10.10.10.1.64141 > s3-1-w.amazonaws.com.http: Flags [S], seq 2601035886, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 768497473 ecr 0,sackOK,eol], length 0

17:40:10.294845 CF +QoS IP 10.10.10.1.64141 > s3-1-w.amazonaws.com.http: Flags [S], seq 2601035886, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 768497473 ecr 0,sackOK,eol], length 0

User accessing cloud-based storage
Countermeasures Against Eavesdropping
Encrypt channels

Solution we’ve seen before:

\[ \{X\}_k \]

Getting the protocols right is another matter…
ssh session

$ ssh wscheirer@140.247.178.71

# tcpdump -X -n tcp port 22

11:28:41.937021 IP 140.247.178.71.22 > 140.247.178.194.48111: Flags [P.], seq 1338:1386, ack 1458, win 247, options [nop,nop,TS val 1250596981 ecr 4256522663], length 48

After protocol exchanges, data packets are encrypted
ssh tunneling

Local port forwarding:

```bash
ssh -L 8080:www.server.org:80 <host>
```

Remote port forwarding:

```bash
ssh -R 5900:localhost:5900 guest@walter-pc
```

Pros: Secure connect through a firewall to use SMTP, IMAP and WWW services

Cons: Internal users can open internal services up to the world

https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding
Application Layer Encryption

Apply encryption here

Data

UDP header

UDP data

IP header

IP data

Frame header

Frame data

Frame footer

Application

Transport

Internet

Link
Secure Socket Layer (SSL)

Two purposes of this protocol:

1. Provide a confidentiality pipe between a browser and a web server
2. Authenticate the server, and possibly the client

Combines several cryptographic facets we discussed in Unit 2
SSL Handshake with Certs.

RN_c = Random number from client
RN_s = Random number from server
SSL Handshake with Certs.

Public key client  🥇 Public key server
Private key client  🥈 Private key server

server certificate
-demand client certificate
-check server certificate

Phase 2

Schematic representation of the SSL handshake protocol with two way authentication with certificates. © BY-SA 3.0 Christian Friedrich
SSL Handshake with Certs.

Schematic representation of the SSL handshake protocol with two way authentication with certificates. © BY-SA 3.0 Christian Friedrich

Public key client (Client) → Private key client

Public key server (Server) → Private key server

client certificate → check client certificate → client certificate (encrypted with Private key client)

check encrypted client certificate → generate random number RNc → send (PMS) encrypted with Private key server

calculate Master-Secret with (PMS, RNc, RNs) → send (MS) encrypted with Private key server

PMS = Pre-Master-Secret  MS = Master-Secret
SSL Handshake with Certs.

Schematic representation of the SSL handshake protocol with two way authentication with certificates. CC BY-SA 3.0 Christian Friedrich
Transport Layer Security (TLS)

- Successor to SSL
- If you need application-specific encryption, use version 1.2 or newer
TLS 1.2 enhancements

RFC 5246

• The MD5-SHA-1 combination in the pseudorandom function (PRF) replaced with SHA-256, with an option to use cipher suite specified PRFs.

• The MD5-SHA-1 combination in the finished message hash replaced with SHA-256, with an option to use cipher suite specific hash algorithms.

• The MD5-SHA-1 combination in the digitally signed element replaced with a single hash negotiated during handshake, which defaults to SHA-1.

• Enhancement in the client's and server's ability to specify which hash and signature algorithms they will accept.

• Expansion of support for authenticated encryption ciphers, used mainly for Galois/Counter Mode (GCM) and CCM mode of Advanced Encryption Standard encryption.

• TLS Extensions definition and AES cipher suites were added.
IPSEC

General network-layer encryption

- Encrypts each IP packet of the session

Transport Mode:

Tunnel Mode:

Only payload is encrypted

Entire IP packet is encrypted and encapsulated into a new IP packet

IPsec transport and tunnel mode © BY 3.0 Ford prefect
Authentication Headers

- Guarantees connectionless integrity and data origin authentication of IP packets
- Protects against replay attacks
- Operates directly on top of IP, using IP protocol 51

![Diagram showing IP header, AH header, Original IP header, TCP header, TCP payload / data]

Protects payload and all non-mutable IP header fields
Encapsulating Security Payloads

Tunnel Mode: Entire IP Packet is encapsulated in a new IP packet

Should be used in conjunction with authentication header

Encrypted original packet

<table>
<thead>
<tr>
<th>IP header</th>
<th>ESP</th>
<th>Encapsulated IP packet or TCP header + payload</th>
<th>ESP</th>
</tr>
</thead>
</table>

Security Associations (SA)

Establishment of shared security attributes between two network entities

Framework for establishing security associations