CSE 40567 / 60567: Computer Security

Network Security 1
Homework #5 is due tonight at 11:59PM (your timezone)

See Assignments Page on the course website for details
Live Tuesday in Class
RC Johnson (PayPal)

(Will not be recorded)
Course Roadmap

Basics
(weeks 1 & 2)

The Web
(weeks 15 & 16)

3 Core Areas

(weeks 3 - 6)

(weeks 6 - 10)

(weeks 11 - 15)
Heartbleed

What is it? Serious input validation vulnerability in OpenSSL

Bug introduced: OpenSSL version 1.0.1 on March 14, 2012

Bug disclosed: April 7th, 2014

Protocol affected: TLS
- Successor to SSL
- Meant to secure network traffic from eavesdropping attacks
- Good example of security software making things less secure

http://heartbleed.com/
Heartbleed bug explained

Heartbeat – Normal usage

Client

Server, send me this 4 letter word if you are there: "bird"

bird

Server

Heartbeat – Malicious usage

Client

Server, send me this 500 letter word if you are there: "bird"

bird. Server master key is 31431498531054. User Carol wants to change password to "password 123"

Server

A depiction of Heartbleed  BY-SA 3.0 FenixFeather
The bug in OpenSSL

\( p \) is a pointer to start of message

```c
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```

No bounds checking to enforce consistency between `payload_length` and the actual payload

The fix:

```c
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```
Scope of the vulnerability
Response from the community

"OpenSSL is not developed by a responsible team."

- Theo de Raadt

Large mistake: OpenSSL developers wrote their own memory management routines, circumventing library protections

"We are building the most important technologies for the global economy on shockingly underfunded infrastructure."

- Dan Kaminsky

An often overlooked downside of the Open Source movement
Fork of OpenSSL by OpenBSD Project
Removed 90,000 lines of C code in its first week

**Key changes:**
Replaced custom memory calls with ones in the standard library
  - Enables ASLR, use of the NX bit, stack canaries
Planning for the future
  - Enables compiler flag checks, fixes for 2038 compatibility
Proper use of cryptographic primitives
Removal of old code and insecure features
Brief Review of TCP/IP
Protocols

The basic protocols that underly the Internet are very mature

1983: IPv4 deployed to ARPANET
1998: IPv6 formalized by IETF
  ▸ Still waiting for universal adoption

Most work in networking (and new security vulnerabilities) now stems from emerging applications
Operation of the TCP/IP Protocol Suite
Protocol layers

Encapsulation of application data descending through the layered IP architecture

© BY-SA 3.0 Cburnett
## ARP Header

<table>
<thead>
<tr>
<th>octet offset</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Hardware type (HTYPE)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Protocol type (PTYPE)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Hardware address length (HLEN)</td>
<td>Protocol address length (PLEN)</td>
</tr>
<tr>
<td>6</td>
<td>Operation (OPER)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Sender hardware address (SHA) (first 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>(next 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>(last 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Sender protocol address (SPA) (first 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>(last 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Target hardware address (THA) (first 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>(next 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>(last 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Target protocol address (TPA) (first 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>(last 2 bytes)</td>
<td></td>
</tr>
</tbody>
</table>
# IPv4 Header

## IPv4 Header Format

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Version</td>
<td>IHL</td>
<td>DSCP</td>
<td>ECN</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>Identification</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>Time To Live</td>
<td>Protocol</td>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td>Source IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>128</td>
<td>Destination IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>160</td>
<td>Options (if IHL &gt; 5)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IPv6 Header

Diagram of the IPv6 packet header. © BY-SA 3.0 Helix84 and HorsePunchKid
TCP Header

<table>
<thead>
<tr>
<th>Data Offset</th>
<th>Reserved</th>
<th>URG</th>
<th>ACK</th>
<th>PSH</th>
<th>RST</th>
<th>SYN</th>
<th>FIN</th>
<th>Window Size</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
</table>

Sequence Number

Acknowledgement Number

Checksum

Urgent Pointer

Options

Data

20 Bytes
TCP Connections

Client

Server

\textit{syn} seq=x

\textit{syn ack}=x+1 seq=y

ack=y+1 seq=x+1

[data]
TCP Connections

Client

- ESTABLISHED (connection)
  - active close
    - FIN_WAIT_1

- FIN_WAIT_2

- TIME_WAIT

- CLOSED

Server

- ESTABLISHED (connection)
  - CLOSE_WAIT
  - passive close
  - LAST_ACK

- CLOSED
UDP Header

Source Port  Destination Port
UDP Length    UDP Checksum
Data

8 Bytes
UDP Request / Response Regime
ICMP Header

<table>
<thead>
<tr>
<th>8-bit ICMP Type</th>
<th>8-bit ICMP Code</th>
<th>16-bit ICMP Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ICMP Contents (dependent on type and code)</td>
</tr>
</tbody>
</table>
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


...
tcpdump

Targeted sniffing is often more useful.

Capture packets to a particular destination:
   tcpdump -n dst host 192.168.1.1

Capture packets from a particular source:
   tcpdump -n src host 192.168.1.1

Capture packets to / from a particular host:
   tcpdump -n host 192.168.1.1

Capture packets to / from a particular network:
   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=ci6ojYCjvFMJM8zqU8gwyYKk88yr0B9iSpOAcwKEk5k2NgLm7IOAcO6pw9iJxbbb8w9MsET2p-J-i0V0b2VBdvnTU8H6XZI1qqw6dT9ZxwcXw9-Tb8EvcyTYQLRUiesU6_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

```bash
# 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
       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
       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

User-friendly front-end for packet sniffing
Wireshark

![Wireshark screenshot](image)
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);