Packet Inspection Praktikum

Martin Brunner, Jonas Pfoh
martin.brunner@sit.fraunhofer.de; pfoh@sec.in.tum.de

IT Security Lab
Technische Universität München
Munich, Germany

Sept. 6, 2010 - Sept. 24, 2010
Outline

1. Introduction
2. Submission Requirements
3. Background
4. Networking Primer
5. libpcap
6. Literature
Welcome

**Goal:** Build a tool that will recognize shellcode within packets.

You will learn:

1. the layout and formation of IP packets
2. programming with the pcap library
3. what shellcode is and how it works
4. how shellcode can be recognized
**Goal:** Build a tool that will recognize shellcode within packets.

You will learn:

1. the layout and formation of IP packets
2. programming with the pcap library
3. what shellcode is and how it works
4. how shellcode can be recognized
Curriculum

- **Sept. 6** - Introduction, Course requirements, libpcap introduction, Milestone 1 assigned
- **Sept. 7 - 8** - Work on Milestone 1
- **Sept. 9** - Milestone 1 due, Milestone 2 assigned
- **Sept. 10** - Work on Milestone 2
- **Sept. 13** - Milestone 2 due, Introduction to buffer overflows, shellcode, & shellcode detection, Milestone 3 & 4 assigned
- **Sept. 14 - 16** - Work on Milestone 3 & 4
- **Sept. 17** - Milestone 3 due
- **Sept. 20 - 23** - Work on Milestone 4
- **Sept. 24** - Milestone 4 due, Final demonstration
Curriculum

- **Sept. 6** - Introduction, Course requirements, libpcap introduction, Milestone 1 assigned
- **Sept. 7 - 8** - Work on Milestone 1
- **Sept. 9** - Milestone 1 due, Milestone 2 assigned
- **Sept. 10** - Work on Milestone 2
- **Sept. 13** - Milestone 2 due, Introduction to buffer overflows, shellcode, & shellcode detection, Milestone 3 & 4 assigned
- **Sept. 14 - 16** - Work on Milestone 3 & 4
- **Sept. 17** - Milestone 3 due
- **Sept. 20 - 23** - Work on Milestone 4
- **Sept. 24** - Milestone 4 due, Final demonstration
Curriculum

- **Sept. 6** - Introduction, Course requirements, libpcap introduction, Milestone 1 assigned
- **Sept. 7 - 8** - Work on Milestone 1
- **Sept. 9** - Milestone 1 due, Milestone 2 assigned
- **Sept. 10** - Work on Milestone 2
- **Sept. 13** - Milestone 2 due, Introduction to buffer overflows, shellcode, & shellcode detection, Milestone 3 & 4 assigned
- **Sept. 14 - 16** - Work on Milestone 3 & 4
- **Sept. 17** - Milestone 3 due
- **Sept. 20 - 23** - Work on Milestone 4
- **Sept. 24** - Milestone 4 due, Final demonstration
Curriculum

- **Sept. 6** - Introduction, Course requirements, libpcap introduction, Milestone 1 assigned
- **Sept. 7 - 8** - Work on Milestone 1
- **Sept. 9** - Milestone 1 due, Milestone 2 assigned
- **Sept. 10** - Work on Milestone 2
- **Sept. 13** - Milestone 2 due, Introduction to buffer overflows, shellcode, & shellcode detection, Milestone 3 & 4 assigned
- **Sept. 14 - 16** - Work on Milestone 3 & 4
- **Sept. 17** - Milestone 3 due
- **Sept. 20 - 23** - Work on Milestone 4
- **Sept. 24** - Milestone 4 due, Final demonstration
Milestones

1. tcpdump clone (libpcap)
2. Simple signature detector (extend MS 1)
3. 1-2 page write-up explaining and defending your approach to shellcode detection
4. Shellcode detection tool (extend MS 2)
Submission Requirements

All programming milestones...

- *must* be done in C.
- *must* include a makefile and compile with 'make all'.
- *must* compile on the workstations in the lab.
- *must* make use of the libpcap library.
Submission Requirements

All programming milestones...

- *must* be done in C.
- *must* include a makefile and compile with ’make all’.
- *must* compile on the workstations in the lab.
- *must* make use of the libpcap library.
Submission Requirements

All programming milestones...

- *must* be done in C.
- *must* include a makefile and compile with 'make all'.
- *must* compile on the workstations in the lab.
- *must* make use of the libpcap library.
Submission Requirements

All programming milestones...

- *must* be done in C.
- *must* include a makefile and compile with ’make all’.
- *must* compile on the workstations in the lab.
- *must* make use of the libpcap library.
make & Makefiles

What is 'make'?

What is a 'Makefile'?
make & Makefiles

What is 'make'?
- 'make' is a program that looks for a Makefile in the current working directory and compiles a program based on this Makefile.

What is a 'Makefile'?
What is 'make'?

- 'make' is a program that looks for a Makefile in the current working directory and compiles a program based on this Makefile.

What is a 'Makefile'?

- A Makefile is a file that generally resides in the top-level directory of your source code and specifies how the program should be compiled.
Listing 1: Makefile

```makefile
CC=gcc
CFLAGS=−c −Wall −g
LDFLAGS=−g −lpcap
OBJS=pack.o

all: pack

pack: ${OBJS}
    ${CC} ${LDFLAGS} ${OBJS} −o $@

%.o: %.c
    ${CC} ${CFLAGS} $<

clean:
    rm −rf *.o pack

Key: $@ = target, $< = dependencies, % = match

Use: make all
```
Listing 2: Makefile

CC=gcc
CFLAGS=−c −Wall −g
LDFLAGS=−g −lp pcap
OBJS=pack.o

all: pack

pack: ${OBJS}
    ${CC} ${LDFLAGS} ${OBJS} −o $@

%.o: %.c
    ${CC} ${CFLAGS} $<

clean:
    rm −rf *.o pack

Key: $@ = target, $< = dependencies, % = match

Use: make all
getopt parser

Use getopt\(^1\) to parse commandline arguments!

Important details:

- must include 'unistd.h'
- char * optarg - the option argument (if required)
- int optopt - stores option char in case of error.
- int optind - stores index on next element in argv to be processed.
- int getopt (int argc, char **argv, const char *options) - returns the next option character ('?' on error) and stores the argument in optarg. Options are passed as single characters optionally followed by ':' if they require an argument or '::' if they have an optional argument.

\(^1\)http://www.gnu.org/s/libc/manual/html_node/Getopt.html
getopt parser - example

```c
aflag = 0;

while ((c = getopt (argc, argv, "ab:")) != −1){
    switch (c) {
    case 'a':
        aflag = 1;
        break;
    case 'b':
        bvalue = optarg;
        break;
    case '?':
        if (optopt == 'b')
            printf (stderr, "Option --%c requires an argument.\n", optopt);
        else if (isprint (optopt))
            printf (stderr, "Unknown option --%c.\n", optopt);
        else
            printf (stderr, "Unknown option character '\\x%lx'.\n", optopt);
        return 1;
        default:
            abort ();
    }
}
printf ("aflag = %d, bvalue = %s\n", aflag, bvalue);
for (index = optind; index < argc; index++)
    printf ("Non-option argument %s\n", argv[index]);
```
Debugging

Don’t rely on `printf`, debuggers are your friend! Learn to use them.

Recommendations:

- **gdb** - traditional/powerful commandline debugger
- **ddd** - front-end GUI for gdb, nice for beginners
- **valgrind** - excellent for tracking down causes of segfaults and debugging memory leaks
Common Ether Types:

- IPv4 (0x0800)
- IPv6 (0x86DD)
- ARP (0x0806)

\[2\text{http://www.shrani.si/f/2P/K3/1gUBV9rc/700px-ethernettypeiifram.png}\]
IPv4 (Layer 3 - Network)

IPv4 (Layer 3 - Network)
TCP (Layer 4 - Transport)


TCP Flags:
- C: Congestion Window
- E: 0x80 Reduced (CWR)
- 0x40 ECN Echo (ECE)
- U: 0x20 Urgent
- A: 0x10 Ack
- P: 0x08 Push
- R: 0x04 Reset
- S: 0x02 Syn
- F: 0x01 Fin

Pending State (DSB, ECN bits):
- Syn: 00 11
- Syn-Ack: 00 01
- Ack: 01 00

Possible Options:
- End of Options List
- No Operation (NOP, Pad)
- Maximum segment size
- Window Scale
- Selective ACK ok
- Timestamp

Checksum: Sum of entire TCP segment and pseudo header (parts of IP header)

RFC 793:
Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

Copyright 2004 - Matt Baxter - mjo@fatpipe.org

---

Length refers to length of entire datagram in bytes.

ICMP (Layer 4 - Transport)

ICMP Message Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Code/Name</th>
<th>Type</th>
<th>Code/Name</th>
<th>Type</th>
<th>Code/Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Echo Reply</td>
<td>3</td>
<td>Destination Unreachable</td>
<td>11</td>
<td>Time Exceeded</td>
</tr>
<tr>
<td>1</td>
<td>Net Unreachable</td>
<td>12</td>
<td>Host Unreachable for TOS</td>
<td>0</td>
<td>TTL Exceeded</td>
</tr>
<tr>
<td>2</td>
<td>Host Unreachable</td>
<td>13</td>
<td>Communication Administratively Prohibited</td>
<td>1</td>
<td>Fragment Reassembly Time Exceeded</td>
</tr>
<tr>
<td>3</td>
<td>Protocol Unreachable</td>
<td>4</td>
<td>Source Quench</td>
<td>0</td>
<td>Parameter Problem</td>
</tr>
<tr>
<td>4</td>
<td>Port Unreachable</td>
<td>5</td>
<td>Redirect</td>
<td>1</td>
<td>Pointer Problem</td>
</tr>
<tr>
<td>5</td>
<td>Fragmentation required, and DF set</td>
<td>6</td>
<td>Redirect Datagram for the Network</td>
<td>1</td>
<td>Missing a Required Operand</td>
</tr>
<tr>
<td>6</td>
<td>Source Route Failed</td>
<td>7</td>
<td>Redirect Datagram for the Host</td>
<td>2</td>
<td>Bad Length</td>
</tr>
<tr>
<td>7</td>
<td>Destination Network Unknown</td>
<td>8</td>
<td>Redirect Datagram for the TOS &amp; Network</td>
<td>3</td>
<td>Timestamp</td>
</tr>
<tr>
<td>8</td>
<td>Destination Host Unknown</td>
<td>9</td>
<td>Redirect Datagram for the TOS &amp; Host</td>
<td>13</td>
<td>Timestamp Reply</td>
</tr>
<tr>
<td>9</td>
<td>Source Host Isolated</td>
<td>10</td>
<td>Router Advertisement</td>
<td>14</td>
<td>Information Request</td>
</tr>
<tr>
<td>10</td>
<td>Network Adminstratively Prohibited</td>
<td>11</td>
<td>Router Advertisement</td>
<td>15</td>
<td>Information Reply</td>
</tr>
<tr>
<td>11</td>
<td>Host Adminstrively Prohibited</td>
<td>12</td>
<td>Router Advertisement</td>
<td>16</td>
<td>Information Reply</td>
</tr>
<tr>
<td>12</td>
<td>Network Unreachable for TOS</td>
<td>13</td>
<td>Router Advertisement</td>
<td>17</td>
<td>Address Mask Request</td>
</tr>
</tbody>
</table>

Checksum

- Checksum of ICMP header
- RFC 792

Please refer to RFC 792 for the Internet Control Message protocol (ICMP) specification.

Network Byte Order

What is byte order (or endianness)?
Network Byte Order

What is byte order (or endianness)?
- The order in which bytes are stored within a system.

What is the byte order used in IP? in x86?
Network Byte Order

What is byte order (or endianness)?

- The order in which bytes are stored within a system.

What is the byte order used in IP? in x86?

- IP = big endian byte order, x86 = little endian byte order

How do we solve this problem?
Network Byte Order

What is byte order (or endianness)?
- The order in which bytes are stored within a system.

What is the byte order used in IP? in x86?
- IP = big endian byte order, x86 = little endian byte order

How do we solve this problem?
- htons(), htonl(), ntohs(), ntohl() (include 'netinet/in.h')
libpcap Usage

1. Open source (device or file)
   - pcap_open_offline
   - pcap_open_live

2. Set filter
   - pcap_lookupnet
   - pcap_compile
   - pcap_setfilter

3. Set callback function
   - pcap_loop

4. Handle incoming packets

5. Close Source
   - pcap_close
libpcap Usage

1. Open source (device or file)
   - `pcap_open_offline`
   - `pcap_open_live`

2. Set filter
   - `pcap_lookupnet`
   - `pcap_compile`
   - `pcap_setfilter`

3. Set callback function
   - `pcap_loop`

4. Handle incoming packets
5. Close Source
   - `pcap_close`
libpcap Usage

1. Open source (device or file)
   - `pcap_open_offline`
   - `pcap_open_live`

2. Set filter
   - `pcap_lookupnet`
   - `pcap_compile`
   - `pcap_setfilter`

3. Set callback function
   - `pcap_loop`

4. Handle incoming packets

5. Close Source
   - `pcap_close`
libpcap Usage

1. Open source (device or file)
   - pcap_open_offline
   - pcap_open_live

2. Set filter
   - pcap_lookupnet
   - pcap_compile
   - pcap_setfilter

3. Set callback function
   - pcap_loop

4. Handle incoming packets

5. Close Source
   - pcap_close
libpcap Usage

1. Open source (device or file)
   - pcap_open_offline
   - pcap_open_live

2. Set filter
   - pcap_lookupnet
   - pcap_compile
   - pcap_setfilter

3. Set callback function
   - pcap_loop

4. Handle incoming packets

5. Close Source
   - pcap_close
libpcap Hints

- Include 'pcap.h' and compile with '-lpcap'
- Protocol header structs already exist for the most part in various headers
- Read:
  - http://www.tcpdump.org/pcap.htm
- Use man page as reference
  (http://www.tcpdump.org/pcap3_man.html)
Literature

1. **getopt**

2. **libpcap**
   - http://www.tcpdump.org/pcap.htm (read)
   - http://www.tcpdump.org/pcap3_man.html (ref)

3. **Buffer Overflows/Shellcode**
   - www.hdm-stuttgart.de/~ms096/haking9-shell2_DE.pdf (read)
   - http://insecure.org/stf/smashstack.html (read)