libpcap

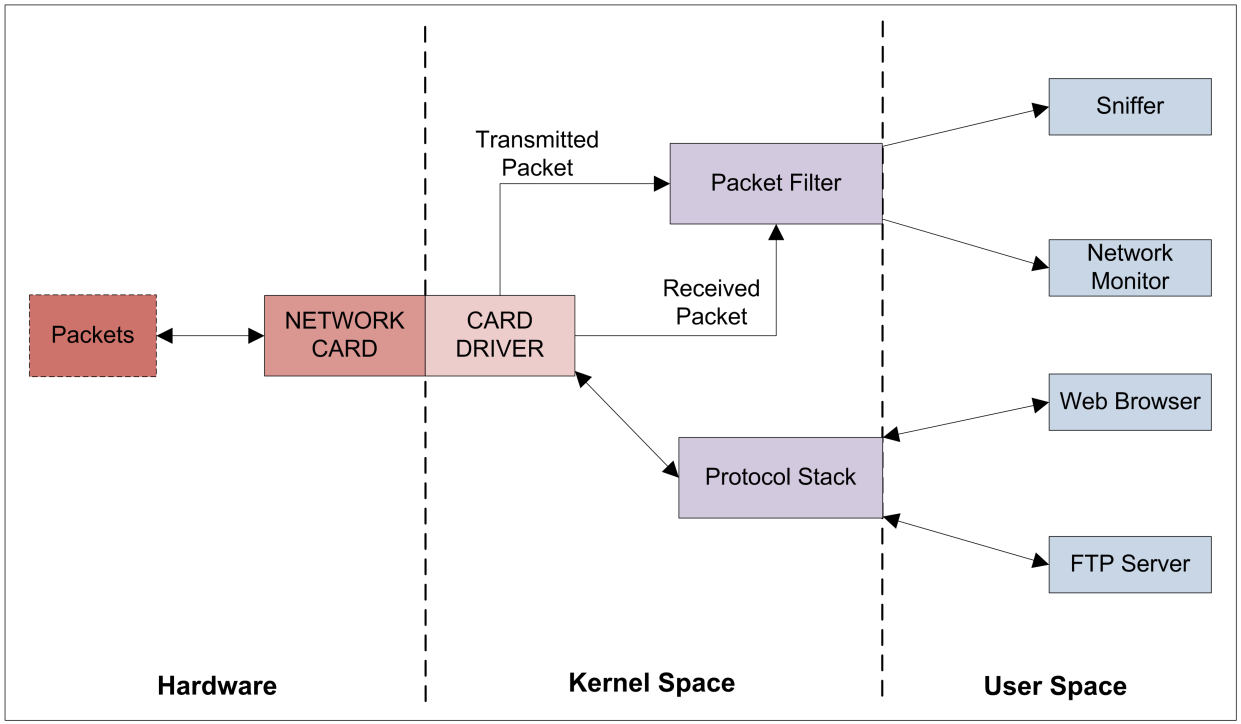
Tématicky zaměřený vývoj aplikací v jazyce C
skupina Systémové programování – Linux

Jiří Novosad

Fakulta informatiky
Masarykova univerzita
novosad@fi.muni.cz

Brno, 1. 12. 2015

libpcap



- packet capture knihovna (C/C++)
- zpracovává data ještě před OS (včetně NetFilteru!)
- obvykle vyžaduje práva superuživatele
- dostupná i pro Windows (WinPcap)
- informace viz `man pcap`, www.tcpdump.org/

Aplikace

Aplikace používající libpcap

tcpdump

- CLI packet analyzátor
- www.tcpdump.org/

tcpdump

- CLI packet analyzátor
- www.tcpdump.org/

úkol

- kdo je přihlášen na Nymfě, pustí v terminálu příkaz `hostname | mail -s $USER novosad@fi.muni.cz`
- vyzkoušejte si `tcpdump -D`
- vyberte si interface a vyzkoušejte `tcpdump -ni eth0`

tcpdump

- CLI packet analyzátor
- www.tcpdump.org/

úkol

- kdo je přihlášen na Nymfě, pustí v terminálu příkaz `hostname | mail -s $USER novosad@fi.muni.cz`
- vyzkoušejte si `tcpdump -D`
- vyberte si interface a vyzkoušejte `tcpdump -ni eth0`
- použijte filtr a uložte DNS komunikaci `resolveip`

Wireshark

- grafická obdoba tcpdumpu
- www.wireshark.org/

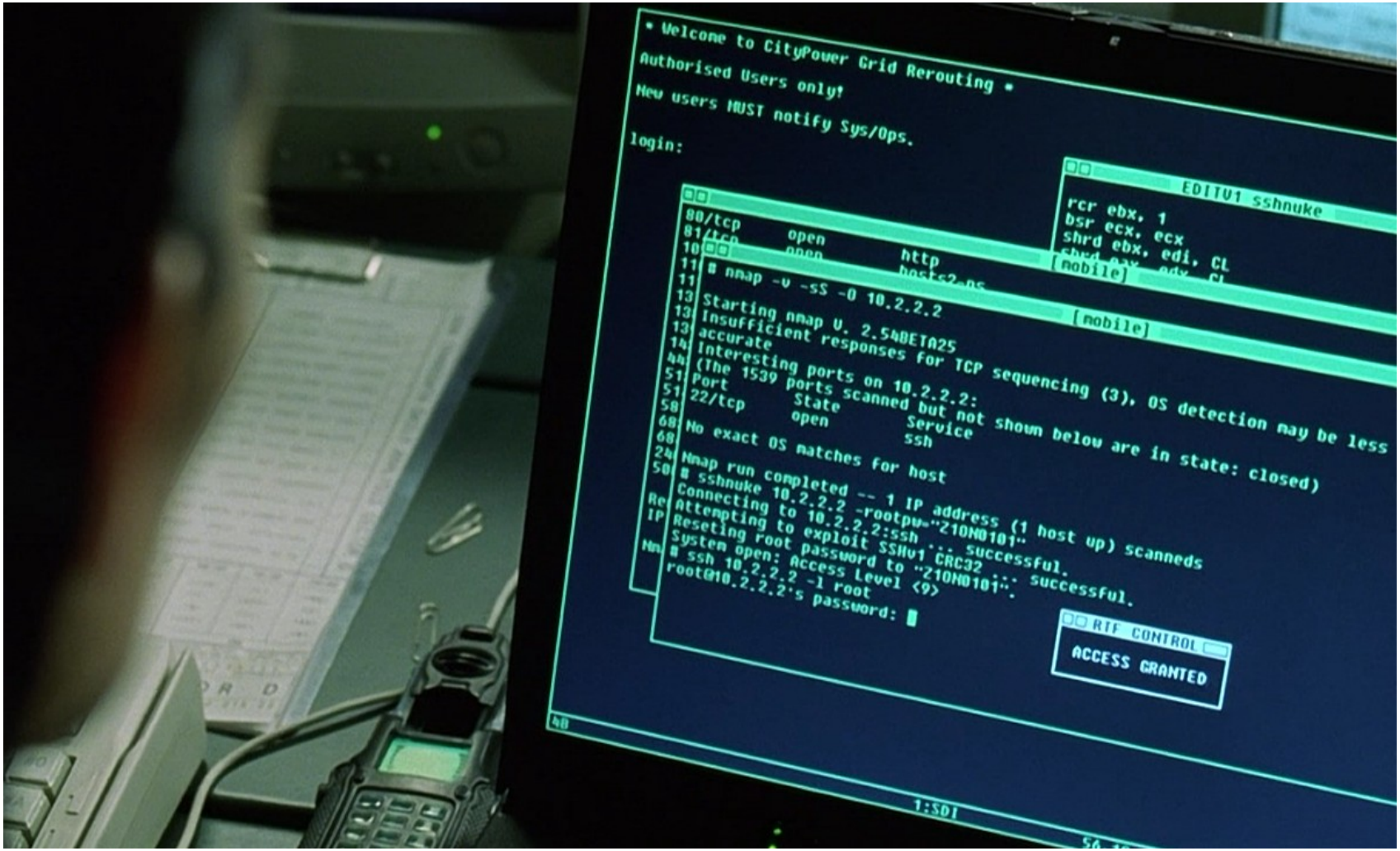
úkol

- otevřete vzorek uložený tcpdumpem a analyzujte provoz

Další aplikace

- snort, suricata (IDS)
- Bro (IDS)
- ngrep (network grep)
- ...

Nmap (port scanner)



API

libpcap API

Specifikace zařízení

- pcapu je třeba říct, z kterého síťového zařízení má brát data

```
#include <pcap/pcap.h>

char errbuf[PCAP_ERRBUF_SIZE];

int pcap_findalldevs(pcap_if_t **alldevsp, char *errbuf);
void pcap_freealldevs(pcap_if_t *alldevs);
char *pcap_lookupdev(char *errbuf);
```

Nymfe: Makefile

```
SOURCE = devlist.c
CFLAGS = -Wall -std=c99 -D_GNU_SOURCE

.PHONY: clean run install all

all: run

bin: $(SOURCE)
        $(CC) $(CFLAGS) $< -lpcap -o bin

install: bin
        @cp bin /var/tmp/pcap/bin

run: install
        @sleep 0.1
        /var/tmp/pcap/bin eth0

clean:
        rm -f bin
```

úkol

- Vytiskněte informace o všech síťových interfacech:
  - jméno
  - popis
  - aktuálně přidělené IPv4 a IPv6 adresy (pokud jsou)

Capture handler

- 2 režimy: online a offline

```
#include <pcap/pcap.h>
pcap_t *pcap_open_live(const char *device, int snaplen,
        int promisc, int to_ms, char *errbuf);

pcap_t *pcap_open_offline(const char *fname, char *errbuf);
pcap_t *pcap_fopen_offline(FILE *fp, char *errbuf);

pcap_t *pcap_open_dead(int linktype, int snaplen);

void pcap_close(pcap_t *p);

char *pcap_geterr(pcap_t *p);
void pcap_perror(pcap_t *p, char *prefix);
```

Capture handler

- 2 režimy: online a offline

```
#include <pcap/pcap.h>
pcap_t *pcap_open_live(const char *device, int snaplen,
        int promisc, int to_ms, char *errbuf);

pcap_t *pcap_open_offline(const char *fname, char *errbuf);
pcap_t *pcap_fopen_offline(FILE *fp, char *errbuf);

pcap_t *pcap_open_dead(int linktype, int snaplen);

void pcap_close(pcap_t *p);

char *pcap_geterr(pcap_t *p);
void pcap_perror(pcap_t *p, char *prefix);

int pcap_datalink(pcap_t *p);
DLT_EN10MB
int pcap_list_datalinks(pcap_t *p, int **dlt_buf);
void pcap_free_datalinks(int *dlt_list);
const char *pcap_datalink_val_to_name(int dlt);
const char *pcap_datalink_val_to_description(int dlt);
```

Zpracování paketů

- cyklus pro zpracování každého paketu
- buď is napíšete vlastní, nebo použijete cyklus pcapu

Zpracování paketů

- cyklus pro zpracování každého paketu
- buď is napíšete vlastní, nebo použijete cyklus pcapu

```
int pcap_next_ex(pcap_t *p, struct pcap_pkthdr **pkt_header,
                 const u_char **pkt_data);
const u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h);

struct pcap_pkthdr {
    struct timeval ts;      /* time stamp */
    bpf_u_int32 caplen;     /* length of the captured part */
    bpf_u_int32 len;        /* length of packet (on wire) */
}
```

Zpracování paketů

- cyklus pro zpracování každého paketu
- buď is napíšete vlastní, nebo použijete cyklus pcapu

```
int pcap_next_ex(pcap_t *p, struct pcap_pkthdr **pkt_header,
                 const u_char **pkt_data);
const u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h);

struct pcap_pkthdr {
    struct timeval ts;      /* time stamp */
    bpf_u_int32 caplen;     /* length of the captured part */
    bpf_u_int32 len;        /* length of packet (on wire) */
}

int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user);
void pcap_breakloop(pcap_t *);

void (*pcap_handler)(u_char *user, const struct pcap_pkthdr *h,
    const u_char *bytes);

int pcap_get_selectable_fd(pcap_t *p);
int pcap_setnonblock(pcap_t *p, int nonblock, char *errbuf);
```

úkol

- Vytvořte počítadlo paketů
- Vstupním parametrem je název síťového zařízení
- Program počítá pakety procházející zařízením
- Při ukončení (Ctrl+C) je vytištěn počet paketů

Filtry

- BPF (BSD Packet Filter) formát
- biot.com/capstats/bpf.html
- `man pcap-filter`

Filtry

- BPF (BSD Packet Filter) formát
- biot.com/capstats/bpf.html
- `man pcap-filter`

```
int pcap_compile(pcap_t *p, struct bpf_program *fp,
        const char *str, int optimize, bpf_u_int32 netmask);
void pcap_freecode(struct bpf_program *);

int pcap_setfilter(pcap_t *p, struct bpf_program *fp);


int pcap_lookupnet(const char *device, bpf_u_int32 *netp,
                bpf_u_int32 *maskp, char *errbuf);
PCAP_NETMASK_UNKNOWN
```

Vyzkoušejte: `tcpdump -d ip and tcp dst port 7`

Závěr

goto

domácí úkoly a zdroje

GOTO

```
int main(int argc, char **argv) {
    int retVal = 0;

    char *memBuf = (char *) malloc(256 * 1024);
    if (NULL == memBuf) {
        fprintf(stderr, "malloc() failed.\n"); retVal = -1;
        goto cleanup_generic;
    }

    FILE *fp = fopen("dummy-file.txt", "w");
    if (NULL == fp) {
        fprintf(stderr, "fopen() failed.\n"); retVal = -2;
        goto cleanup_malloc;
    }

    int sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (-1 == sock) {
        fprintf(stderr, "socket() failed.\n"); retVal = -3;
        goto cleanup_file;
    }

    printf("--> Perform some normal operations, sing a song, ...\n");
...
```

GOTO cont.

```
...
    int sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (-1 == sock) {
        fprintf(stderr, "socket() failed.\n"); retVal = -3;
        goto cleanup_file;
    }

    printf("--> Perform some normal operations, sing a song, ...\n");

cleanup_socket:
    close(sock);
cleanup_file:
    fclose(fp);
cleanup_malloc:
    free(memBuf);
cleanup_generic:
    return retVal;
}
```

Domácí úkol

Síťový analyzátor

- Volání: `./sniff eth0`
- Z IP paketů (v Ethernetu, `DLT_EN10MB`) zjistěte následující informace
  - čas přijetí paketu
  - L2: src, dst MAC adresa
  - L3 (IPv4 i IPv6):
    - src, dst adresa
    - TTL (IPv6: hop limit)
    - protokol (UDP/TCP)
  - L4 (UDP i TCP): src, dst port
- Informace vypisujte v reálném čase
- Zachycujte jen IPv4/IPv6 pakety (použijte správný filtr)
- Typ: `netinet/{ether,ip,ip6,tcp}.h`, `ether_ntoa()`, `inet_ntop()`, `ntohs()`

Zdroje

- www.tcpdump.org/
- www.wireshark.org/
- biot.com/capstats/bpf.html
- www.tcpdump.org/linktypes.html
- blog.cloudflare.com/bpf-the-forgotten-bytecode/
- eli.thegreenplace.net/2009/04/27/using-goto-for-error-handling-in-c