

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Dokumentácia
Projekt 2 - zadanie ZETA
IPK 2019/2020

Obsah

1	Úvod	2
2	Implementácia	3
2.1	Preklad	3
2.2	Používané knižnice	3
2.3	Program	3
3	Testovanie	6
4	Bibliografia	7

1 Úvod

Úlohou tohto projektu bolo naprogramovať zachytávač TCP a UDP paketov, ktorý má možnosť filtrovať zachytené pakety – určiť sledovaný port, sledovaný protokol a počet zachytených paketov. Príkaz na spustenie:

```
1 ./ipk-sniffer -i rozhranie [-p port] [--tcp|-t] [--udp|-u] [-n num]
```

Zachytené pakety sú vypisované na štandardný výstup (*STDOUT*) v nasledujúcej forme:

```
1 20:44:42.271769 fra15s22-in-f3.1e100.net : 80 > student-vm : 37870
2
3 0x0000: 08 00 27 c5 c2 16 52 54 00 12 35 02 08 00 45 00 ..'...RT..5...E.
4 0x0010: 02 36 13 cd 00 00 40 06 94 6a ac d9 17 a3 0a 00 .6....@..j.....
5 0x0020: 02 0f 00 50 93 ee 02 5a 8a 02 c9 60 a8 5e 50 18 ...P...Z...`^P.
6 0x0030: ff ff d3 35 00 00 ...5..
7
8 0x0040: 48 54 54 50 2f 31 2e 31 20 33 30 31 20 4d 6f 76 HTTP/1.1 301 Mov
9 0x0050: 65 64 20 50 65 72 6d 61 6e 65 6e 74 6c 79 0d 0a ed Permanently..
10 0x0060: 4c 6f 63 61 74 69 6f 6e 3a 20 68 74 74 70 3a 2f Location: http:/
11 0x0070: 2f 77 77 77 2e 67 6f 6f 67 6c 65 2e 73 6b 2f 0d /www.google.sk/.
12 0x0080: 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 .Content-Type: t
13 0x0090: 65 78 74 2f 68 74 6d 6c 3b 20 63 68 61 72 73 65 ext/html; charse
14 0x0100: 74 3d 55 54 46 2d 38 0d 0a 44 61 74 65 3a 20 53 t=UTF-8..Date: S
15 0x0110: 75 6e 2c 20 30 33 20 4d 61 79 20 32 30 32 30 20 un, 03 May 2020
16 0x0120: 31 38 3a 34 34 3a 34 33 20 47 4d 54 0d 0a 45 78 18:44:43 GMT..Ex
17 0x0130: 70 69 72 65 73 3a 20 54 75 65 2c 20 30 32 20 4a pires: Tue, 02 J
18 0x0140: 75 6e 20 32 30 32 30 20 31 38 3a 34 34 3a 34 33 un 2020 18:44:43
19 0x0150: 20 47 4d 54 0d 0a 43 61 63 68 65 2d 43 6f 6e 74 GMT..Cache-Cont
20 0x0160: 72 6f 6c 3a 20 70 75 62 6c 69 63 2c 20 6d 61 78 rol: public, max
21 0x0170: 2d 61 67 65 3d 32 35 39 32 30 30 30 0d 0a 53 65 -age=2592000..Se
22 0x0180: 72 76 65 72 3a 20 67 77 73 0d 0a 43 6f 6e 74 65 rver: gws..Conte
23 0x0190: 6e 74 2d 4c 65 6e 67 74 68 3a 20 32 31 38 0d 0a nt-Length: 218..
24 0x0200: 58 2d 58 53 53 2d 50 72 6f 74 65 63 74 69 6f 6e X-XSS-Protection
25 0x0210: 3a 20 30 0d 0a 58 2d 46 72 61 6d 65 2d 4f 70 74 : 0..X-Frame-Opt
26 0x0220: 69 6f 6e 73 3a 20 53 41 4d 45 4f 52 49 47 49 4e ions: SAMEORIGIN
27 0x0230: 0d 0a 0d 0a 3c 48 54 4d 4c 3e 3c 48 45 41 44 3e ....<HTML><HEAD>
28 0x0240: 3c 6d 65 74 61 20 68 74 74 70 2d 65 71 75 69 76 <meta http-equiv
29 0x0250: 3d 22 63 6f 6e 74 65 6e 74 2d 74 79 70 65 22 20 ="content-type"
30 0x0260: 63 6f 6e 74 65 6e 74 3d 22 74 65 78 74 2f 68 74 content="text/ht
31 0x0270: 6d 6c 3b 63 68 61 72 73 65 74 3d 75 74 66 2d 38 ml; charset=utf-8
32 0x0280: 22 3e 0a 3c 54 49 54 4c 45 3e 33 30 31 20 4d 6f ">.<TITLE>301 Mo
33 0x0290: 76 65 64 3c 2f 54 49 54 4c 45 3e 3c 2f 48 45 41 ved</TITLE></HEA
34 0x0300: 44 3e 3c 42 4f 44 59 3e 0a 3c 48 31 3e 33 30 31 D><BODY>.<H1>301
35 0x0310: 20 4d 6f 76 65 64 3c 2f 48 31 3e 0a 54 68 65 20 Moved</H1>.The
36 0x0320: 64 6f 63 75 6d 65 6e 74 20 68 61 73 20 6d 6f 76 document has mov
37 0x0330: 65 64 0a 3c 41 20 48 52 45 46 3d 22 68 74 74 70 ed.<A HREF="http
38 0x0340: 3a 2f 2f 77 77 77 2e 67 6f 6f 67 6c 65 2e 73 6b ://www.google.sk
39 0x0350: 2f 22 3e 68 65 72 65 3c 2f 41 3e 2e 0d 0a 3c 2f /">here</A>...</
40 0x0360: 42 4f 44 59 3e 3c 2f 48 54 4d 4c 3e 0d 0a BODY></HTML>..
```

Prvý odsek obsahuje čas zachytenia paketu, adresu odosielateľa a port, adresu prijímateľa a port. Druhý odsek obsahuje hlavičku paketu a tretí obsah (*payload*) zobrazené vo forme: offset, charaktery v hexadecimálnej podobe a charaktery v ASCII podobe. Nezobraziteľné znaky sú zobrazené ako . (bodka).

Zoznam odovzdaných súborov: src/main.cpp, Makefile, manual.pdf, README

2 Implementácia

Zadanie projektu som implementoval v jazyku C++ písaného v štandarde C++11.

2.1 Preklad

Program sa prekladá pomocou nástroja **make** spusteného v koreni priečinku:

```
1 make
```

Makefile spúšťa príkaz:

```
1 g++ -Wall -std=c++11 -o ipk-sniffer src/main.cpp -lpcap
```

Vytvára sa spustiteľný súbor **ipk-sniffer**.

2.2 Použité knižnice

```
1 #include <iostream>
2 #include <string>
3 #include <cstring>
4 #include <ctime>
5 #include <chrono>
6 #include <atomic>
7 #include <iomanip>
8
9 #include <unistd.h>
10 #include <sys/types.h>
11 #include <sys/socket.h>
12 #include <getopt.h>
13
14 #include <net/ethernet.h>
15 #include <netdb.h>
16 #include <netinet/if_ether.h>
17 #include <netinet/in.h>
18 #include <netinet/ip.h>
19 #include <netinet/ip6.h>
20 #include <netinet/ip_icmp.h>
21 #include <netinet/tcp.h>
22 #include <netinet/udp.h>
23 #include <arpa/inet.h>
24 #include <pcap.h>
```

Knižnice z posledného odseku sú použité pre definície štruktúr ako ip header, atď. (**netinet**) a **pcap.h** pre funkcie na zachytávanie paketov.

2.3 Program

Pri spustení programu sa inicializujú globálne a lokálne (vo funkcii main) premenné, načítajú sa argumenty a podľa nich program ďalej pokračuje. Ak zadáme správne argumenty (argument -i s rozhraním, na ktorom chceme zachytávať pakety), program pokračuje a načíta všetky dostupné rozhrania, cez ktoré potom postupne prechádza a porovnáva ich názov s rozhraním, ktoré zadal používateľ:

```
1 pcap_if_t* device_list = nullptr;
2 if(pcap_findalldevs(&device_list, errbuf) != 0) {
3     cout << "pcap_findalldevs() failed:" << errbuf << endl;
4     return 1;
5 }
```

```

6   pcap_if_t* device = nullptr;
7   for (pcap_if_t* curr_device = device_list; curr_device; curr_device = curr_device
->next) {
8       if (show_interfaces == true)          {cout << curr_device->name << endl;}
9       if (curr_device->name == interface) {device = curr_device;}
10  }

```

Ak rozhranie existuje, pokúsi sa ho otvoriť a následne začne sledovať pakety:

```

1   handle = pcap_open_live(device->name, BUFSIZ, 1, 0, errbuf);
2   if (!handle) {
3       pcap_freealldevs(device_list);
4       cout << "pcap_open_live() failed: " << errbuf << endl;
5       return 1;
6   }
7   pcap_freealldevs(device_list);
8
9   if (pcap_loop(handle, -1, callbackPacketHandler, nullptr) == PCAP_ERROR) {
10      pcap_close(handle);
11      cout << "pcap_loop() failed: " << pcap_geterr(handle);
12      return 1;
13  }
14  pcap_close(handle);

```

Pokiaľ sme program spustili aj s argumentom **-p**, tak sa skompiluje a nastaví filter na port:

```

1   if (port != "") {
2       bpf_program filter;
3       if (pcap_compile(handle, &filter, port.c_str(), 1, 0) == -1) {
4           pcap_close(handle);
5           cout << "pcap_compile() failed: " << pcap_geterr(handle);
6           return 1;
7       }
8       if (pcap_setfilter(handle, &filter) == -1) {
9           pcap_close(handle);
10          cout << "pcap_setfilter() failed: " << pcap_geterr(handle);
11          return 1;
12      }
13  }

```

V tele programu som využil funkcie práve z knižnice **pcap.h**.

Funkcia **pcap_loop()** volá takzvanú *callback* funkciu, kde sa odohráva logika za získaním informácií z paketu a ich vypísanie. Názorná ukážka časti tejto funkcie pre TCP pakety:

```
1 void callbackPacketHandler(u_char *args, const struct pcap_pkthdr* header, const
  u_char* packet_buffer) {
2     if (packetCount <= 0) {pcap_breakloop(handle);}
3
4     auto time = saveTime();
5     auto size = header->len; //size of a whole packet
6     const struct iphdr* ip_header = (struct iphdr*)(packet_buffer + sizeof(struct
  ethhdr));
7     u_short ip_header_len = (ip_header->ihl) * 4;
8
9     auto version = AF_INET;
10    if (ip_header->version == 6) {version = AF_INET6;}
11
12    struct in_addr addr_src_bin; struct in_addr addr_dest_bin;
13    addr_src_bin.s_addr = ip_header->saddr;
14    addr_dest_bin.s_addr = ip_header->daddr;
15    auto addr_src = inet_ntoa(addr_src_bin);
16    auto addr_dest = inet_ntoa(addr_dest_bin);
17    struct hostent* name_src; struct hostent* name_dest;
18
19    switch (ip_header->protocol)
20    {
21        case 6: { // TCP
22            if ((packetCount <= 0) or (mode != "both" and mode != "tcp")) {break;}
23
24            const struct tcphdr* tcp_header { (struct tcphdr*)(packet_buffer + sizeof(
  struct ethhdr) + ip_header_len)};
25            auto header_size = sizeof(struct ethhdr) + ip_header_len + (tcp_header->
  doff) * 4;
26            const u_char* data = packet_buffer + header_size;
27            auto data_size = size - header_size;
28
29            printTime(time);
30
31            if ((name_src = gethostbyaddr(&addr_src_bin, sizeof(addr_src_bin),
  version)) != nullptr) {
32                cout << name_src->h_name;
33            } else {cout << addr_src;}
34
35            cout << " : " << ntohs(tcp_header->source) << " > ";
36
37            if ((name_dest = gethostbyaddr(&addr_dest_bin, sizeof(addr_dest_bin),
  version)) != nullptr) {
38                cout << name_dest->h_name;
39            } else {cout << addr_dest;}
40
41            cout << " : " << ntohs(tcp_header->dest) << endl << endl;
42            printPacket(packet_buffer, header_size, data, data_size);
43
44            --packetCount;
45            break;
46        }
47        ...

```

3 Testovanie

Svoj program som testoval pomocou programov **Firefox**, **Curl** a **Wireshark**. Pomocou Firefoxu a Curlu som tvoril prevádzku paketov, pomocou svojho programu a Wiresharku som ju zachytával a výsledky porovnával.

The screenshot displays the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The main packet list table shows the following data:

No.	Time	Source	Destination	Protocol	Length	Info
...	46.061387912	10.0.2.15	10.0.2.2	SSH	162	Server: Encrypted packet (len=108)
...	46.061626515	10.0.2.2	10.0.2.15	TCP	60	51030 → 22 [ACK] Seq=1697 Ack=1217 Win=65535 Len=0
...	46.459864616	10.0.2.15	77.75.75.1...	TCP	74	53220 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK
...	46.493912580	77.75.75.1...	10.0.2.15	TCP	60	80 → 53220 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MS
...	46.493941553	10.0.2.15	77.75.75.1...	TCP	54	53220 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
+	46.494040275	10.0.2.15	77.75.75.1...	HTTP	127	GET / HTTP/1.1
...	46.494308080	77.75.75.1...	10.0.2.15	TCP	60	80 → 53220 [ACK] Seq=1 Ack=74 Win=65535 Len=0
...	46.546404377	77.75.75.1...	10.0.2.15	HTTP	382	HTTP/1.1 302 Moved Temporarily (text/html)
...	46.546420094	10.0.2.15	77.75.75.1...	TCP	54	53220 → 80 [ACK] Seq=74 Ack=329 Win=63912 Len=0
...	46.546600414	10.0.2.15	77.75.75.1...	TCP	54	53220 → 80 [FIN, ACK] Seq=74 Ack=329 Win=63912 Len=0
...	46.546799485	77.75.75.1...	10.0.2.15	TCP	60	80 → 53220 [ACK] Seq=329 Ack=75 Win=65535 Len=0
...	46.586331984	77.75.75.1...	10.0.2.15	TCP	60	80 → 53220 [FIN, ACK] Seq=329 Ack=75 Win=65535 Len=0
...	46.586350541	10.0.2.15	77.75.75.1...	TCP	54	53220 → 80 [ACK] Seq=75 Ack=330 Win=63912 Len=0
...	47.675049228	10.0.2.15	10.0.2.2	SSH	90	Server: Encrypted packet (len=36)

The packet details pane for the selected HTTP packet (Frame 88) shows the following structure:

- Frame 88: 127 bytes on wire (1016 bits), 127 bytes captured (1016 bits) on interface 0
- Ethernet II, Src: PcsCompu_c5:c2:16 (08:00:27:c5:c2:16), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
- Internet Protocol Version 4, Src: 10.0.2.15, Dst: 77.75.75.176
- Transmission Control Protocol, Src Port: 53220, Dst Port: 80, Seq: 1, Ack: 1, Len: 73
- Hypertext Transfer Protocol

The packet bytes pane shows the raw data in hexadecimal and ASCII. The ASCII column shows the following text:

```
RT..5...E.  
q..@..MK  
K...P..t..P  
m..GE T / HTTP  
/1.1..Ho st: sezn  
am.cz..U ser-Agen  
t: curl/ 7.58.0..  
Accept: /*.*...
```

Below the Wireshark window, a terminal window shows the command 'curl http://seznam.cz' and its output, which is an HTML response with a 302 Found status:

```
student@student-vm: ~  
File Edit View Search Terminal Tabs Help  
student@student-v... student@student-v...  
student@student-vm:~$ curl http://seznam.cz  
<html>  
<head><title>302 Found</title></head>  
<body>  
<center><h1>302 Found</h1></center>  
<hr><center>nginx</center>  
</body>  
</html>
```

4 Bibliografia

Literatúra

- [1] pcap_loop manpage. [online], [vid. 2020-04-25].
URL https://linux.die.net/man/3/pcap_loop
- [2] CARSTENS, T.: Programming with pcap. [online], [vid. 2020-04-25].
URL <https://www.tcpdump.org/pcap.html>
- [3] fffaraz: [online], [vid. 2020-04-25].
URL <https://gist.github.com/fffaraz/7f9971463558e9ea9545>