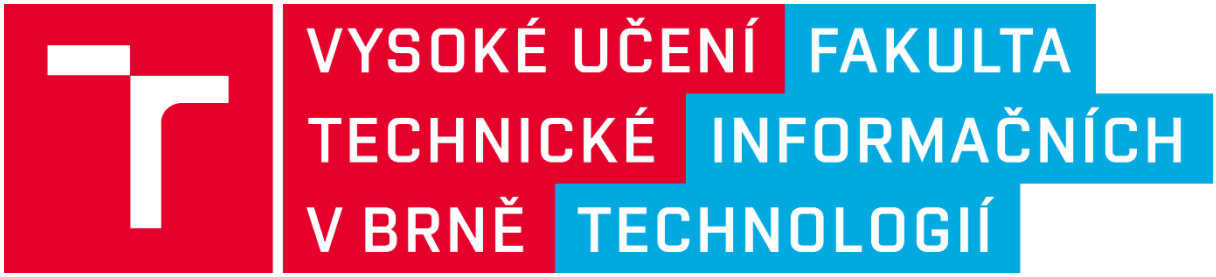

Sniffer packetů

Tomáš Souček

xsouce15

OBSAH

1. Úvod

Tato dokumentace slouží k popisu druhého projektu z předmětu IPK, konkrétně varianty Zeta: sniffer packetů. Sniffer dokáže na základě zadaných argumentů filtrovat podle typu protokolů, portu, rozhraní a požadovaném počtu odchycených packetů. Na standartní výstupní kanál následně vypíše informace o chycených packetech, jako jsou např. MAC a IP adresy příjemce i odesílatele, porty, délky paketů, …

2. Implementace

Projekt byl implementován v jazyce C za použití síťových knihoven. V projektu bylo nejvíce čerpáno z knihovny pcap, která se stará o odchytávání filtrem určených packetů na daném rozhraní.
K překladu zdrojového kódu byl vytvořen soubor Makefile, který používá gcc kompilátor.

2.1 Main

Na začátku proběhne defince a deklarace všech později potřebných proměnných. Následně se začnou zpracovávat argumenty, jako na kterém rozhraní a na kterém portu poslouchat, případně jaký typ packetů filtrovat a kolik packetů má program odchytit. V případě, že nebylo specifikováno rozhraní, vypíšou se všechna dostupná. Pokud bylo specifikováno, díky pcap funkcím se provede jeho otevření a začne se díky promiskuitnímu módu naslouchat.

2.2 Pomocné funkce

processPacket

callback funkce pro pcap_loop(). Funkce dostane ukazatel na ethernet hlavičku a ukazatel na samotný packet. Ze samotná hlavičky následně funkce bere hodnotu velikosti celého packetu v bytech a určuje se timestamp.

U ukazatele na packet proběhne přetypování na ethhdr strukturu, ze které se následně dá zjistit typ přenosové vrstvy (IPv4, IPv6 nebo ARP) a podle něj zavolat funkce pro další zpracování.

Z přetypovaného packetu se ve funkci také zjišťuje MAC adresa příjemce a odesílatele.

processIpv4

Přetypování packetu na iphdr strukturu, která obsahuje potřebné informace o datovém protokolu. Během přetypování se ještě musí ukazatel posunout o offset ethernetové hlavičky, tak, aby ukazatel mířil na začátek dat o IPv4 vrstvě, jinak by došlo k misinterpretaci dat packetu.

Následně se podle typu protokolu rozhodne, zda se následně bude zpracovávat ICMP, TCP nebo UDP část packetu.

processIpv6

Stejně jako u processIpv4 funkce proběhne přetypování a rozhodnutí o typu protokolu. Rozdílné chování oproti IPv4 můžeme vidět ve zjištění a vypsání IP adres odesílatele a příjemce již v této funkci, nikoliv v pozdějších částech programu.

processArp

U ARP rámců jsou vypsané pouze IP adresy odesílatele a příjemce. Jelikož se mi nepodařilo najít vhodnou strukturu na zpracování dat, použil jsem u_char, jakožto poměrně univerzální strukturu.

Jednotlivé offsety, pro pozice IP adres, jsem zjistil pomocí programu Wireshark a vypsal je do kódu jako konstanty.

printTcpPacket
Funkce vypíše obsah TCP části packetu, konkrétně IP adresy, porty a data. Funkce funguje pro IPv4 i IPv6 verze, jen se na základě parametru version rozliší velikost offsetu IP hlavičky. Vytisknutá data jsem zvolil jako celý obsah TCP části v hexadecimální podobě.

printUdpPacket
Funkcionálně se neliší od printTcpPacket. Důvodem pro zvláštní funkci ale byla potřeba použití jiné struktury pro přetypování ukazatele na packet.

printIcmpPacket
U ICMP packetu opět dochází k nastavení ip offsetu na základě verze IP protokolu. Jelikož ale ICMP packet neobsahuje porty, vytiskne se jen IP hlavička, v případě IPv4, a data (opět celý obsah ICMP v hexadecimální podobě).

printIPHeader
Proběhne alokace dvou stringů o fixní délce 16 bytů, do kterých jsou následně nahrané hodnoty IP adres odesílatele i příjemce. Rozpoznání IP adres probíhá pomocí přetypování na strukturu ip s offsetem o ethernetovou hlavičku.

printData
Vypíše výše specifikovaná data packetů v hexadecimální podobě s očíslovanými řádky (obdobně jako Wireshark).

3. Spuštění
./ipk-sniffer [-i *rozhraní* | --interface *rozhraní*] {-p *port*} {[--tcp|-t] [--udp|-u] [--arp] [--icmp] } {-n *num*}

kde

- -i eth0 (právě jedno rozhraní, na kterém se bude poslouchat. Nebude-li tento parametr uveden, či bude-li uvedené jen -i bez hodnoty, vypíše se seznam aktivních rozhraní)
- -p 23 (bude filtrování paketů na daném rozhraní podle portu; nebude-li tento parametr uveden, uvažují se všechny porty)
- -t nebo --tcp (bude zobrazovat pouze TCP pakety)
- -u nebo --udp (bude zobrazovat pouze UDP pakety)
- --icmp (bude zobrazovat pouze ICMPv4 a ICMPv6 pakety)
- --arp (bude zobrazovat pouze ARP rámce)
- -n 10 (určuje počet paketů, které se mají zobrazit, tj. i "dobu" běhu programu; pokud není uvedeno, uvažujte zobrazení pouze jednoho paketu, tedy jakoby -n 1)
- argumenty mohou být v libovolném pořadí

4. Testování
Testování v průběhu vývoje bylo prováděno porovnáváním výstupu programu s daty, které vracel program Wireshark. Pro testování byly použity příkazy ping a curl, případně jejich IPv6 verze.

Ve chvíli, kdy program vracel stejná data, jako Wireshark, byl program otestován ve virtuálním Linuxu, jehož obraz lze najít v zadání.

5. Zdroje

1. How to code a Packet Sniffer in C with Libpcap on Linux - BinaryTides. BinaryTides - Coding, Software, Tech and Reviews [online]. Copyright © 2022 [cit. 15.04.2022]. Dostupné z: https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/
2. Bráníme se odposlechu: promiskuitní režim - Lupa.cz. Lupa.cz - server o českém Internetu [online]. Copyright © 1997 [cit. 15.04.2022]. Dostupné z: https://www.lupa.cz/clanky/branime-se-odposlechu-promiskuitni-rezim/
3. Programming with pcap | TCPDUMP & LIBPCAP. Home | TCPDUMP & LIBPCAP [online]. Copyright © 2010 [cit. 15.04.2022]. Dostupné z: https://www.tcpdump.org/pcap.html