VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií

POČÍTAČOVÉ KOMUNIKACE A SÍTĚ

2019/2020

Projekt 2

Sniffer paketů

Tomáš Julina (xjulin08) Zlín, 1. května 2020

Obsah

1. Úvod

Obecné zadání

Vytvořte komunikující aplikaci podle konkrétní vybrané specifikace obvykle za použití libpcap a/nebo síťové knihovny BSD sockets (pokud není ve variantě zadání uvedeno jinak). Projekt bude vypracován v jazyce C/C++/C#. Individuální zadání specifikuje vlastní referenční systém, pod kterým musí být projekt přeložitelný a spustitelný. Program by však měl být přenositelný.

Zadání varianty ZETA

Navrhněte a implementujte síťový analyzátor v C/C++/C#, který bude schopný na určitém síťovém rozhraním zachytávat a filtrovat pakety.

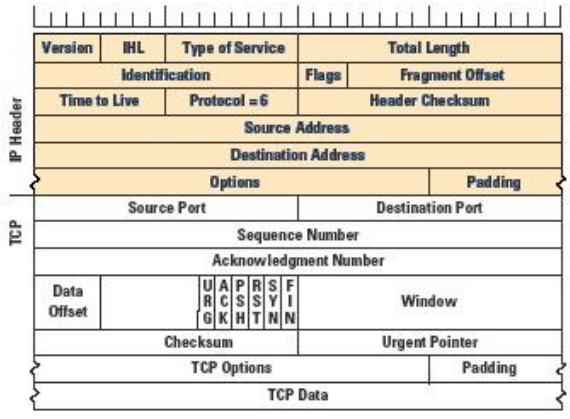
Spuštění programu

./ipk-sniffer -i *rozhraní[,rozhrani2,...]* [-p *port[,port2,...]*] [--tcp|-t] [--udp|-u] [-n *num*] {--help|-h}

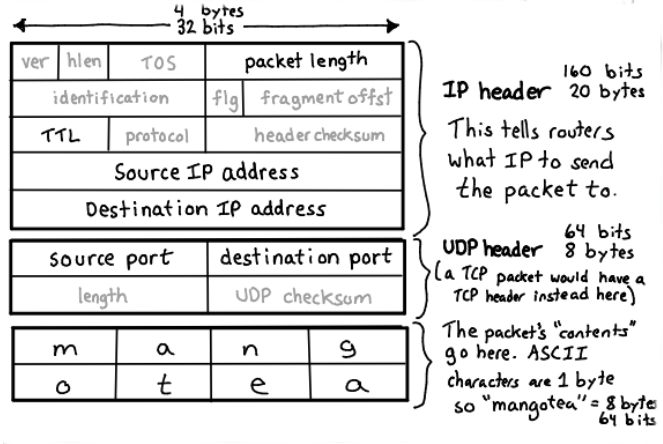
- -i *eth0* (rozhraní, na kterém se bude poslouchat. Nebude-li tento parametr uveden, vypíše se seznam aktivních rozhraní), rozšíření: -i enp0s3,lo (bude poslouchat na více rozhraních zároveň)
- -p *23* (bude filtrování paketů na daném rozhraní podle portu; nebude-li tento parametr uveden, uvažují se všechny porty), rozšíření: -p 23,53 (bude filtrovat na více portech zároveň)
- -t nebo --tcp (bude zobrazovat pouze tcp pakety)
- -u nebo --udp (bude zobrazovat pouze udp pakety), pokud nebude -tcp ani -udp specifikováno, uvažují se TCP a UDP pakety zároveň
- -n *10* (určuje počet paketů, které se mají zobrazit; pokud není uvedeno, uvažujte zobrazení pouze 1 paket, pokud je uvedena hodnota -1, uvažuje se nekonečný výpis paketů – nutné ukončit program manuálně)
- -h nebo --help (vytiskne nápovědu a popis, poté ukončí program)

Co je packet sniffing?

Pokaždé, kdy jsou přes internet přenášena nějaké data, jsou tato data u odesílatele (sender's node) rozdělena na mnoho malých částí = *data packets* a u příjemce (receiver's node) jsou opět poskládány do původního formátu. *Paket* je nejmenší jednotkou komunikace v počítačových sítích a právě odchytávání, následná analýza, filtrace apod. se nazývá *packet sniffing*.



1. TCP/IP paket
https://www.potaroo.net/ispcol/2004-07/fig1.jpg



2. UDP paket
https://jvns.ca/images/packet-headers.png

Základní návrh a princip programu

Program se po spuštění pokusí otevřít určité rozhraní k naslouchání, v případě úspěchu je pokaždé, když se podaří odchytit nějaký paket, volána funkce, která tento paket zpracuje. Paket je struktura, jejíž velikost je možné snadno zjistit, skládá se z více částí a každá část obsahuje určitá data. Na začátku zpracování paketu se zjistí, zda se jedná o TCP či o UDP paket (tato informace je zjistitelná z IP hlavičky) a dále se zjistí, zda odpovídá zadaný port a zda se ještě má další paket vůbec vypsat, v případě, že ano, program paket dále rozdělí a níže popsaným způsobem vypíše na standardní výstup.

2. Implementace

Program byl implementován v jazyce C++, zejména kvůli implementaci rozšíření, ve kterých bylo s výhodou využito datového typu *string*, vektorů z knihovny *vector* a hashovací tabulky z knihovny *map.h.* Pro samotné odchytávání paketů byla využita zejména knihovna *pcap*, při následné zpracovávání paketu pro výpis dle zadaných parametrů byly využity struktury z níže zmíněných zajímavých knihoven.

Zajímavé knihovny

1. *pcap.h* – knihovna byla použita pro odchytávání paketů na vybraném rozhraní [4]
2. *udp.h, tcp.h, ip.h, ip6.h* – knihovny obsahují deklarace hlaviček pro jednotlivé protokoly
3. *getopt.h* – knihovna umožňuje pohodlné zpracování vstupních argumentů
4. *string, map, vector* – knihovny využité k implementaci rozšíření

Zpracování argumentů

Kontrolu a zpracování vstupních argumentů má na starosti funkce *parse_arguments*, argumenty jsou interně zpracovány pomocí funkce *getopt_long*, která umožňuje „dlouhý“ formát přepínačů v kombinaci s klasickým i s krátkým formátem, jednotlivé možnosti jsou vybírány pomocí konstrukce *switch* a v ní jsou nastavovány pomocné proměnné pro řízení dalšího chodu programu, funkce také kontroluje správný tvar, formát a rozsah argumentů. O zpracování portů a rozhraní se starají pomocné funkce *processInterface* a *processPorts*, které pomocí regulárních výrazů zkontrolují formát vstupního řetězce a poté uloží data do vektoru (v případě rozšíření ještě vstupní formát rozdělí podle podporovaného formátu viz. sekce Rozšíření).

Hlavní část programu

V případě, že nebyl zadán argument *-i* s názvem aktivního rozhraní, je využito funkce *pcap_findalldevs*, která nalezne všechny aktivní a zároveň kompatibilní zařízení (s knihovnou pcap) a uloží je do listu rozhraní, každá položka má uložený název, případně popis apod. Následně je uživateli vypsán seznam těchto aktivních zařízení a program je úspěšně ukončen.

V případě, že byl zadán argument *-i* s existujícím, aktivním rozhraním, je volána funkce *pcap_open_live* (zajímavou implementační volbou je zejména argument s hodnotou 200[ms], který zajistí, že pakety nebudou zpracovávány po jednou, ihned po zachycení, ale v uvedené době se

akumuluje více paketů, které poté budou předány a zpracovány naráz – kdyby tato hodnota chyběla, aplikace by se volala pro každý paket, což by mohlo mít za následek zbytečné vytížení operačního systému), která zadané rozhraní otevře pro odchytávání a umožní získat *packet capture descriptor*. Následně je pomocí funkce *prepareFilter* sestaven filtr, který s využitím funkce *pcap_compile* a *pcap_setfilter* pomůže ignorovat nevyhovující pakety. Po aplikování filtrů již zbývá pouze odchytávat pakety v rámci funkce *pcap_dispatch*, která poté, co je odchytnut nějaký paket, umožní uživateli daný paket jakkoliv zpracovat pomocí uživatelsky definované *callback* funkce.
Program podporuje rozhraní, která vytvářejí ethernetové či linux „cooked" caption hlavičky (typ hlaviček linkové vrstvy).

Callback funkce

Tato funkce je volána pokaždé, když se podaří odchytit nějaký paket. Na začátku funkce se dle protokolu zachyceného paketu rozhodne, zda se jedná o TCP či UDP paket (bylo by možné další dělení, ovšem to se po nás v zadání nežádalo). Hlavička (*header*) paketu se skládá z IP hlavičky (obsahuje například informaci o verzi IP, délce této hlavičky, protokolu – pro nás zajímavý TCP a UDP, zdrojovou a cílovou IP adresu apod.), ethernetové hlavičky (popřípadě linux „cooked" capture hlavičky) a TCP/UDP hlavičky (obsahují například zdrojový a cílový port). Z části IP hlavičky je získána zdrojová a cílová adresa, tyto adresy se následně program pomocí funkce *getnameinfo* pokusí přeložit a získat pro ně odpovídající doménové jména, tato jména jsou spolu s odpovídající IP adresou uložena do hashovací tabulky, která je využívána jako *cache* proto, aby se opakovaně nepřekládaly stejné adresy neustále dokola pro každý paket (šetří vytížení sítě a nevytváří zbytečné pakety, které by běžely v cyklu dokola, rozšíření základního zadání), z části UDP respektive TCP hlavičky je naopak získán zdrojový a cílový port a v poslední části paketu lze nalézt datový obsah paketu (pokud nějaký paket obsahuje). Takto získaná data jsou již dle zadání vypsána na standardní výstup pomocí funkce *print_packet* (princip funkce inspirován ze zdroje [11]), lze toho docílit poměrně jednoduchým způsobem – na začátku každého řádku je vytisknut hexadecimálně počet vypsaných bajtů, poté se vypíše šestnáct bajtů obsahu z datového bufferu hexadecimálně a následně ty samé bajty v klasické ASCII podobě, pokud se jedná o netisknutelné znaky, vypíše se tečka (v případě, že by řádek nebyl celý se zbytek doplní mezerami), takto se vypíše počet paketů, dle vstupních argumentů (když není uvedeno, vypíše se jeden paket).
Formát výpisu je následující: na prvním řádku je čas, IP, port zdroje a cíle, následuje prázdný řádek, poté se vypíše zvlášť obsah IP hlavičky, ethernetové hlavičky a TCP (resp. UDP) hlavičky, následuje opět prázdný řádek a v poslední části se vypíše datový obsah paketu (již bez obsahu hlaviček, viz. zadání), takto vypsaný paket je oddělen od následujícího prázdným řádkem.

Rozšíření 1: Odchytávání na více rozhraní, filtrování více portů zároveň

V projektu jsem se rozhodl implementovat možnost odchytávání paketů na více zařízeních současně a také filtrování paketů z více portů současně.

Ukázka formátu rozšíření:

```
sudo ./ipk-sniffer -i lo,enp0s3 -p 80,53
```

Jednotlivá rozhraní i porty od sebe musí být odděleny čárkou, zvažoval jsem, zda u portů nepodporovat i možnost „rozpětí od, do", ovšem vzhledem k povaze paketů jsem se rozhodl, že hledat na více konkrétních portech bude rozumnější.

Aby byla implementace co možná nejčistší s ohledem na již vypracovanou základní verzi, jsem se rozhodl využít vektory a bohatou nabídku práce s řetězci, kterou C++ poskytuje. Při zpracovávání argumentů využívám regulární výrazy (oddělení jednotlivých částí bylo inspirováno [14]), které mi umožní podporovat jak klasickou verzi ze zadání, tak i toto rozšíření, jednotlivé rozhraní a porty jsou ukládány do vektorů a ty jsou dále využívány.

Více rozhraní
Rozhraní v zadaném formátu se zpracuje funkcí *processInterface* (získání jednotlivých názvů z řetězce inspirováno [14]). Každé zadané rozhraní je otevřeno pro odchytávání paketů, poté se využívá funkce *pcap_dispatch*, která vždy zpracovává pakety po „dávkách“, tudíž je možné nerušeně v cyklu while neustále přepínat mezi jednotlivými zařízeními a zpracovávat pakety tam, kde právě nějaké jsou, je využíváno také funkce *pcap_setnonblock*, která umožní tzv. neblokující mód (každá funkce se hned vrátí, tzn. něčeká na příchod paketů a umožní zpracování na dalším rozhraní, díky tomu není nutné využívat více procesů zároveň). Po zpracování a vypsání zadaného množství paketů program skončí.

Více portů
Zpracování z více portů zároveň je umožněno pomocí filtrování paketů s využitím funkcí *pcap_compile* a *pcap_setfilter*, porty jsou po zpracování vstupních dat funkcí *processPorts* (získání jednotlivých portů z řetězce inspirováno [14]) uloženy ve vektoru celých čísel a poté jsou z nich poskládány filtry datového typu *string*, tento filtr potom zařídí to, že pakety s jinými porty než s těmi, které uživatel požadoval, budou ignorovány.

Rozšíření 2: Ukládání přeložených IP adres do cache
Jako druhé rozšíření jsem implementoval ukládání IP adres do cache, což zamezí zbytečnému neustálému posílání DNS paketů s žádostí o *resolving IP* na doménové jméno (což by způsobilo nekonečný cyklus).

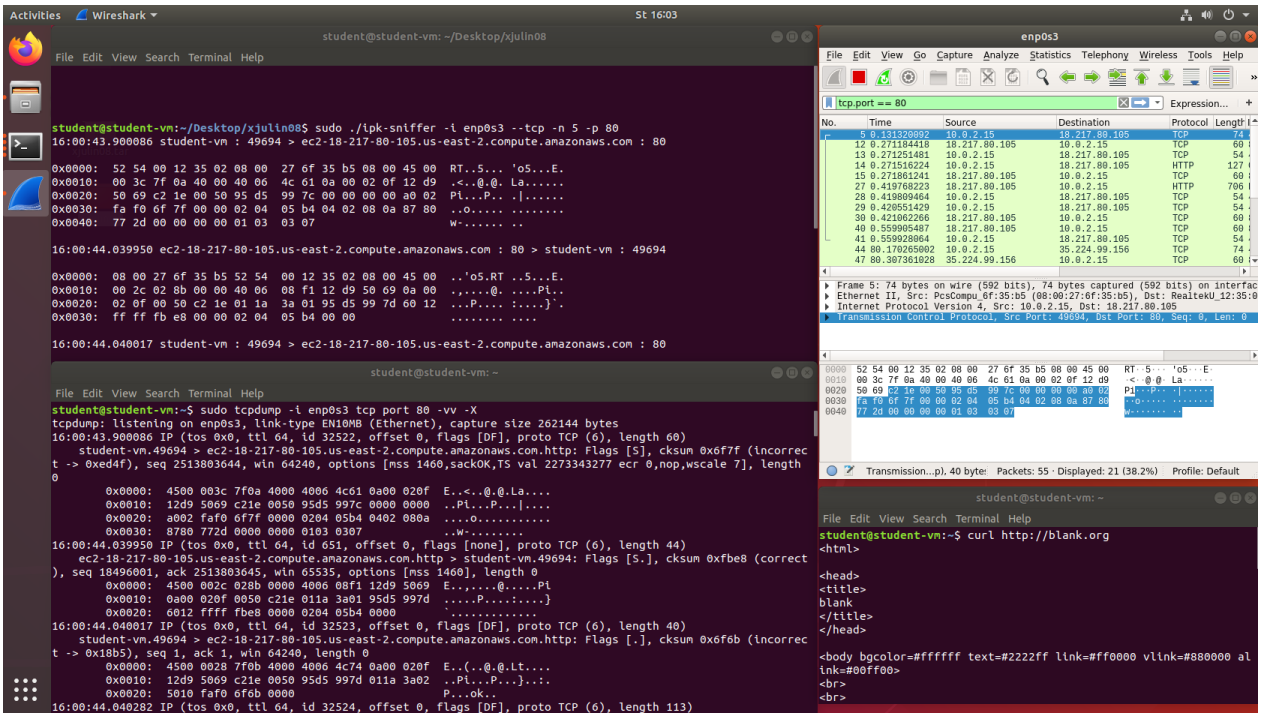
Implementace
Po získání IP adres z paketu jsou tyto adresy vyhledány v hashovací tabulce obsahující IP adresy s již přeloženými doménovými jmény, v případě, že se najde shoda, vytiskne se již uložená hodnota, jinak se požádá o překlad pomocí funkce *getnameinfo*. Ke *kešování* (caching) je využita struktura *std::map* dostupná v jazyce C++, tato struktura slouží jako hashovací tabulka a umožní ukládat data ve formátu *klíč,hodnota*, klíč i hodnota jsou v mém případě datového typu *string*, v klíči se nachází původní IP adresa a hodnota je doménové jméno (popřípadě IP adresa, pokud se překlad nepodařilo provést).

3. Testování

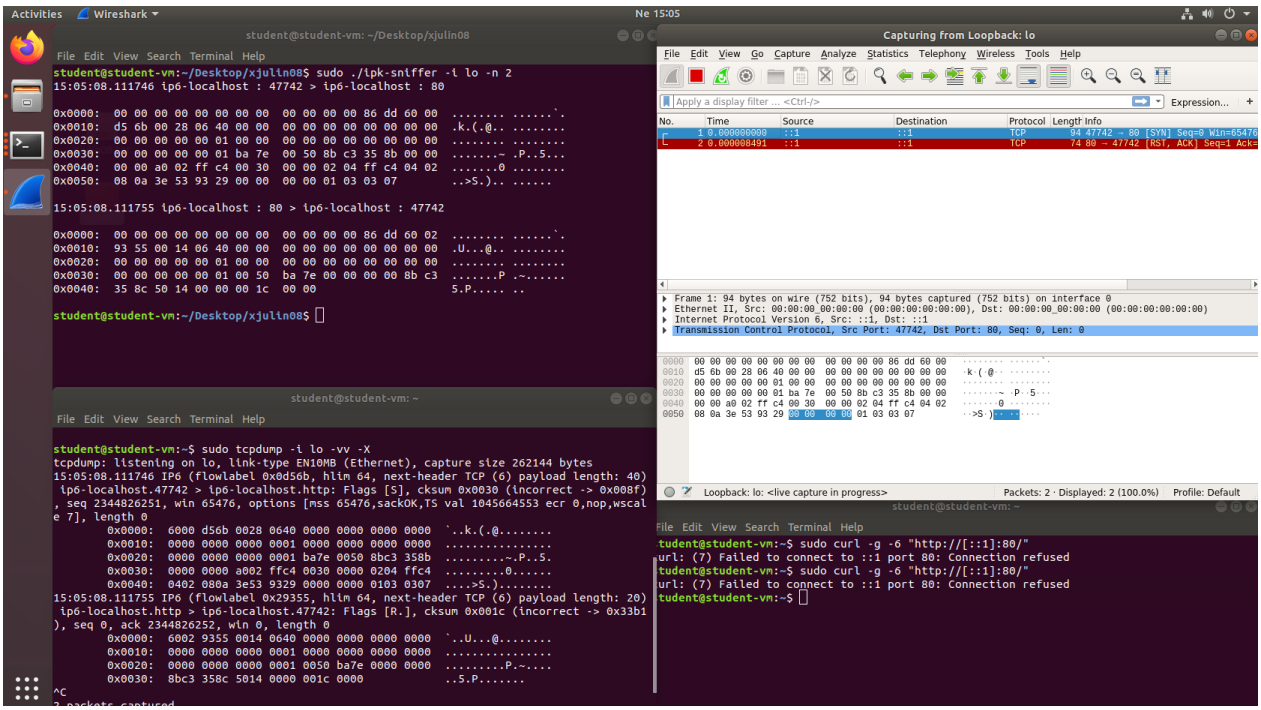
Popis testování
Pro testování jsem se rozhodl využít velice známý (a hlavně neplacený) program Wireshark a TCPdump. Zajímal mě zejména formát informací, které tento program o paketech zobrazí, formát je velice podobný naší projektové variantě, samozřejmě obsahuje více informací (a možností filtrování), ale to je pochopitelné, výsledky byly naprosto srovnatelné s mou implementací. Další testování jsem prováděl již s hotovou implementací na referenčním virtuálním stroji. Pro vytváření

network traffic jsem používal program *curl, ssh* a *Mozilla Firefox*. Při testování jsem se pokoušel odchytávat zejména pakety na portu 80, jelikož se často jedná o protokol http, bylo to pro mě „nejčitelnější“. V průběhu jsem narazil na (pro mě) velice zajímavý případ – a sice, po delším snažení odchytit UDP paket na portu 80 jsem se dočetl, že se tento port ve vztahu „http-over-UDP“ téměř nikdy nepoužívá (v některých případech tohoto portu využívají viry a ostatní malware), ale byl zde i pokus od společnosti Google o vytvoření protokolu (QUIK = Quick UDP Internet Connections), který by zlepšil výkon (čti rychlost, odezvu) webových aplikací na bázi připojení, které doteď využívají protokol TCP.

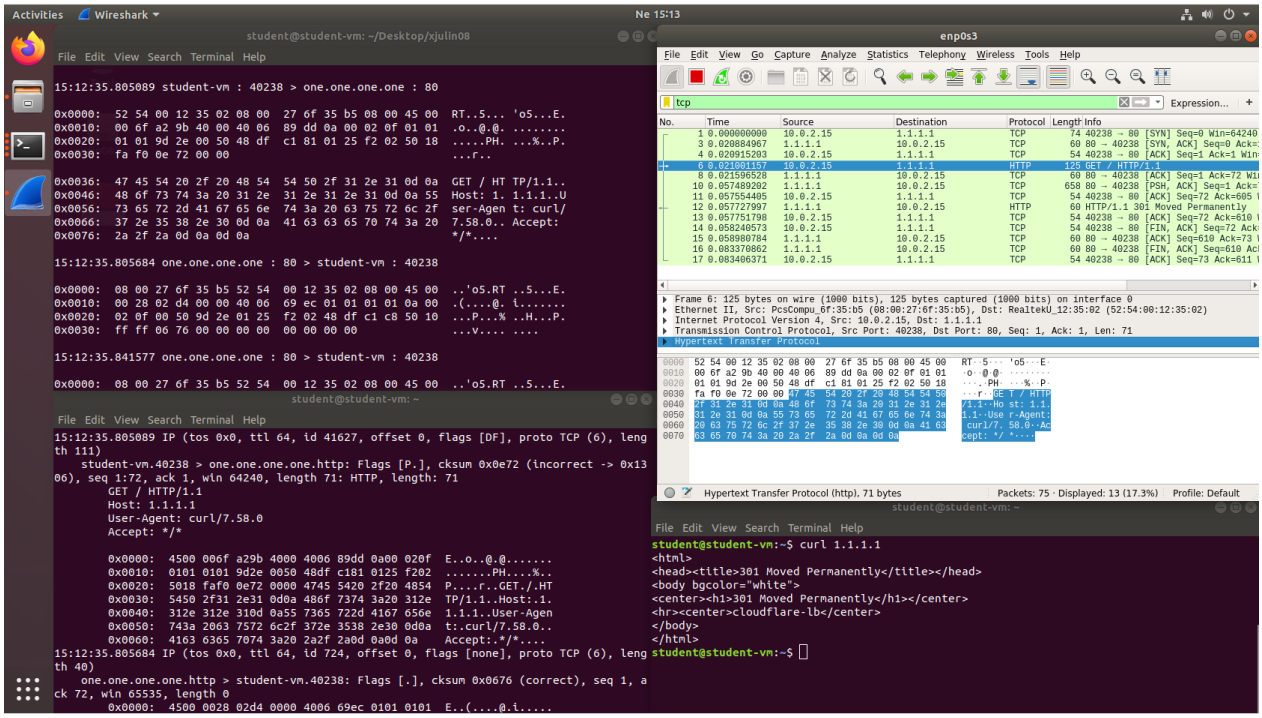
Ukázka testování



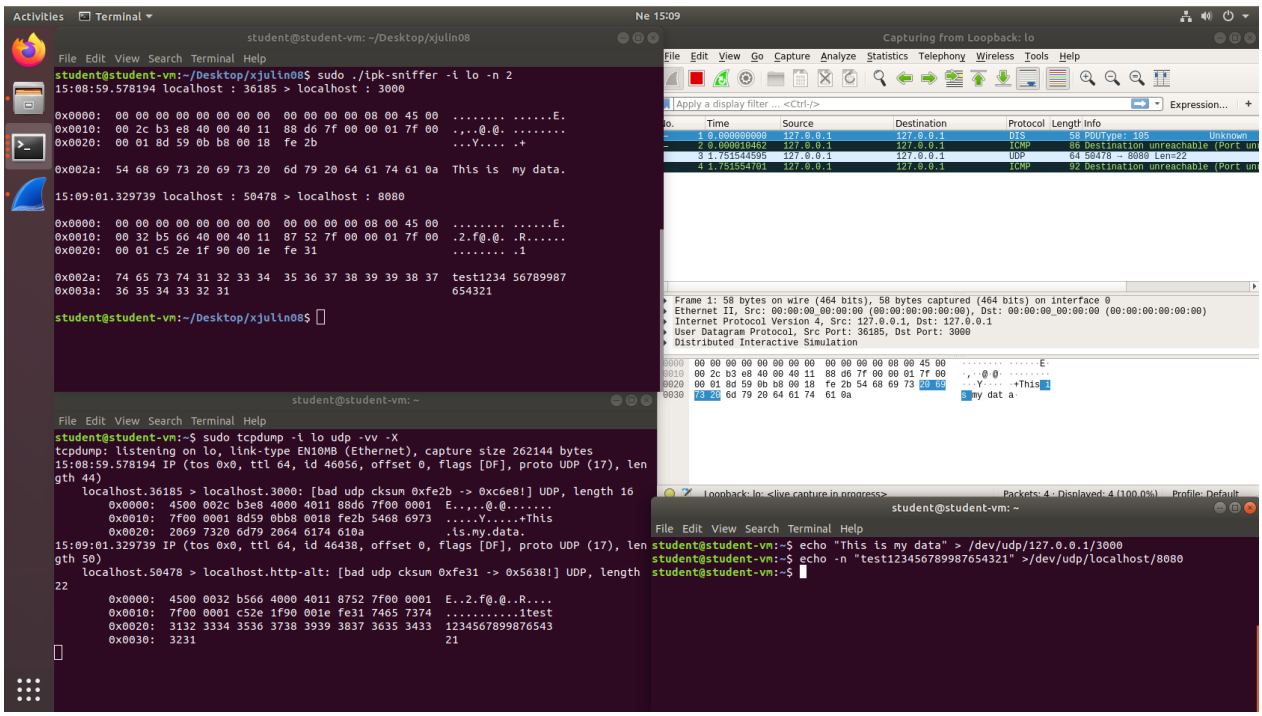
3. Ukázka testování na referenčním stroji, http stránka



4. Ukázka testování na referenčním stroji, IPv6 localhost



5. Ukázka testování na referenčním stroji, one.one.one.one



6. Ukázka testování na referenčním stroji, vlastní paket

4. Zdroje

[1] Paessler.com. *What Is Packet Sniffing? Definition And Details*.
[online] ©2020 Paessler AG [cit. 1.5.2020] Dostupné z: https://www.paessler.com/it-explained/packet-sniffing

[2] Linux.die.net. *Linux Man Pages*. *PCAP, SOCKET, IP, TCP, UDP man pages.*
[online] ©1996, die.net [cit. 1.5.2020] Dostupné z: https://linux.die.net/man

[3] *Wikipedia, the free encyclopedia. Pcap.*
[online] [cit. 1.5.2020] Dostupné z: https://en.wikipedia.org/wiki/Pcap

[4] Tim Carstens. *Programming with pcap*.
[online] ©2020, The Tcpdump Group [cit. 1.5.2020] Dostupné z: http://www.tcpdump.org/pcap.html

[5] Evans, J. *How Big Can A Packet Get?*.
[online] ©Julia Evans. [cit. 1.5.2020] Dostupné z: https://jvns.ca/blog/2017/02/07/mtu/

[6] The Internet Society. *RFC 793 – Transmition Control Protocol Specification*.
[online] ©1981 [cit. 1.5.2020] Dostupné z: https://tools.ietf.org/html/rfc793

[7] The Internet Society. *RFC 791 - Internet Protocol Specification*.
[online] ©1981 [cit. 1.5.2020] Dostupné z: https://tools.ietf.org/html/rfc791

[8] The Internet Society. *RFC 768 - User Datagram Protocol.*
[online] ©1980 [cit. 1.5.2020] Dostupné z: https://tools.ietf.org/html/rfc768

[9] The Internet Society. *RFC 3542 - ADVANCED SOCKETS FOR IPv6*.
[online] ©2003 [cit. 1.5.2020] Dostupné z: https://www.ietf.org/rfc/rfc3542.txt

[10] The Internet Society. *RFC 2460 - Internet Protocol, Version 6 (Ipv6) Specification*.
[online] ©1998 [cit. 1.5.2020] Dostupné z: https://tools.ietf.org/html/rfc2460

[11] Moon, S., 2013. *C Packet Sniffer Code With Libpcap And Linux Sockets*.
[online] ©2020 BinaryTides. Dostupné z: https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/

[12] Mullins, M. *Exploring The Anatomy Of A Data Packet*.
[online] ©2001 TechRepublic. [cit. 1.5.2020] Dostupné z:
https://www.techrepublic.com/article/exploring-the anatomy-of-a-data-packet/

[13] Cplusplus.com. *Cplusplus.Com - The C++ Resources Network*.
[online] ©cplusplus.com, 2000-2020 [cit. 1.5.2020] Dostupné z: http://www.cplusplus.com/

[14] „hayk.mart“. *Parse (Split) A String In C++ Using String Delimiter (Standard C++)*.
In: stackoverflow.com [online] ©2020 Stack Overflow Dostupné z:
https://stackoverflow.com/questions/14265581/parse-split-a-string-in-c-using-string-delimiter-standard-c#comment44856986_14266139