Manual

IPK-sniffer

Počítačové komunikace a sítě

Dokumentace k 2. projektu

Iveta Strnadová
xstrna14

3. května 2020

1. Úvod

Zadání

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

Návrh aplikace

Jazyk a knihovny

Pro implementaci jsem zvolila jazyk C#. Na práci s pakety jsou použity prostředky knihovny *SharpPCap* a *PacketDotNet*. Zpracování argumentů z příkazové řádky probíhá pomocí knihovny *CommandLineParser*.

Struktura programu

Program byl rozdělen do funkcí různých tříd podle cíle a prostředků potřebných pro jejich vykonání. Základem návrhu se staly dvě třídy, *Argument* a *CapturePacket*. Mimo ně by existovala ještě třída *Program* s funkcí *main*, jejímž účelem by bylo pouze vytvořit důležité objekty a zavolat jejich funkce pro naslouchání paketů.

Třída *Argument* byla navrhnuta pro zpracování argumentů programu a uchování důležitých hodnot pro další části. Uvnitř by zapouzdřovala použití knihovny *CommandLineParser* a přidávala vlastní kontroly parametrů a jejich vyhodnocování pro snazší použití později.

Třída *CapturePacket* byla modelována jako jádro programu. Pomocí knihovny *SharpPCap* by se starala o otevření zařízení pro naslouchání i samotné filtrování a zpracování paketů. K filtrování by používala prostředky instance třídy *Argument*.

2. Implementace

Makefile

Pro překlad zdrojových souborů (`make` / `make build`) do výsledné aplikace je použit nástroj *dotnet publish*. Vytvoří spustitelný soubor *ipk-sniffer* pro unixové prostředí (a několik dalších nespustitelných). Obsahuje také příkaz `make clean`, jenž tyto soubory vymaže, a `make run` na spuštění pro zachycení jednoho paketu na zařízení *any*.

ipk-sniffer projekt

Soubor *ipk-sniffer.csproj* obsahuje konfiguraci projektu, kterou *dotnet* využívá při sestavování aplikace.

Implementace všech dále popsaných tříd se nachází v souboru *ipk-sniffer.cs*.

Program

Obsahuje jedinou funkci *Main*. Vytvoří objekt třídy *Argument* i *PacketCapture*. Zavolá metodu na zpracování argumentů z *Argument* a poté objekt předá funkcím na nalezení žádaného zařízení na naslouchání a na zachytávání paketů.

Argument

Hlavním cílem této třídy je zpracovat argumenty z volání programu do vlastních proměnných, ke kterým se později bude moci přistupovat. Používá k tomu *ParseArguments* knihovny *CommandLineParser*. Poté provede dodatečné kontroly a uloží si důležitá data.

Přijímané argumenty

- `-h, --help` Vypíše nápovědu k možným argumentům a stručný popis činnosti programu. Bez ohledu na kombinaci argumentů se program poté ukončí.
- `-i eth0` Určí jméno zařízení, na kterém se má naslouchat. Pouze se uloží jméno, kontrola zda jde o přístupné zařízení je ponechána na třídě *CapturePacket*. Když je nalezen pouze přepínač `-i` bez jména, jedná se o chybu. Pokud není nalezen vůbec, je vypsán seznam aktivních zařízení a program se ukončí.
- `-p port` Číslo portu k naslouchání, buď jako *source port* nebo *destination port*. Pokud není hodnota uvedena, ukládá se informace, že nebude aplikován žádný filtr podle portu. Pokud je uvedena, je nejprve zkontrolována správná hodnota portu (záporná či vyšší než 65535 vede k ukončení programu).
- `-n num` Počet packetů, které má program zachytit a vypsat. Proběhne kontrola, jestli jde o přirozené číslo. V případě hodnoty 0 se ukončí bez chybového návratového kódu. Když tento parametr chybí, bude se zachytávat jeden packet.
- `-u, --udp` / `-t, --tcp` Slouží k filtrování paketů podle jejich protokolu. Nastavením pouze přepínače UDP se nebudou zpracovávat TCP pakety, uvedením pouze TCP se nebudou brát v potaz UDP pakety. Žádný z přepínačů má stejný efekt jako oba dva zároveň - zpracovávají se pakety obou protokolů.

CapturePacket

Největší třída zaměřená na zpracování paketů. Pro větší přehlednost je členěna do více funkcí.

find_device

Slouží k nalezení zařízení na naslouchání. Pomocí *CaptureDeviceList* najde všechny aktivní zařízení a pokusí se mezi nimi najít takové, jehož jméno je stejné jako to extrahované z argumentů. Pokud se jí to podaří, ukládá dané zařízení jako atribut třídy *Device* pro pozdější použití. Když ho nenajde, program končí chybou.

catch_packets

Funkce použije dříve nalezené zařízení *Device* - otevře ho v módu *Promiscuous* pro zachytávání všech paketů, nejen těch co jsou pro program. Dále nastavuje *counter* jako počítadlo sloužící k zpracování žádaného počtu paketů.

Obsahuje cyklus, který v každé své iteraci zachytí paket a pošle na zpracování funkci *work_packet*. (Kdyby byl právě zachycený paket hodnoty *null*, cyklus by se ukončil předčasně. Tato situace by mohla nastat, kdyby došlo k timeout nastaveném na 10s. Jedná se o ochranu před potenciálně nekonečně běžícím programem.)

Pokud funkce *work_packet* vrátí hodnotu *true*, znamená to, že paket prošel filtry programu a byl tudíž zpracován. V takovém případě se přičte 1 k *counter* a porovná se s žádanou hodnotou zpracovaných paketů, pokud je stejná, cyklus se ukončí a zařízení *Device* se uzavře.

work_packet

Z *RawCapture* dat paketu, které jsou funkci předány, je vytvořen *IPPacket* a z něj podle typu *UdpPacket* či *TcpPacket*. Pokud je objeveno, že nejde o ani jeden z protokolů UDP/TCP či jde o takový, který není žádáno zachytávat, funkce končí s návratovou hodnotou *false*.

Funkce dále zpracovává data s cílem vytvořit a vypsat hlavičku tvaru `hh:mm:ss.fff source_IP|FQDN : source_port > destination_IP|FQDN : destination_port`.

Čas je získán z *RawCapture* a jsou z něj převzaty hodiny, minuty, sekundy a milisekundy v UTC. Čas je ponechán v původním čase extrahovaném z paketu, není převáděn do lokálního.

IP adresy source a destination paketu jsou získány z *Tcp/UdpPacket* a program se je pokusí přeložit na doménové jméno. Pokud se překlad podařil, do hlavičky se uvede jméno, pokud skončil s exception, ponechá se IP adresa.

Čísla portů jsou extrahovány z *Tcp/UdpPacket*. Pokud bylo v argumentech omezení na číslo portu, ověří se source i destination hodnota a v případě, že ani jedna neodpovídá, funkce vrací hodnotu *false*.

Do této části se již funkce dostane pouze v případě, že paket prošel všemi filtry. Vypíše se tedy poskládaná hlavička, volá se funkce *parse_packet_body* a poté funkce končí s návratovou hodnotou *true*.

parse_packet_body

Funkce nejprve inicializuje dvě počítadla, *counter* na pohyb v řádku výpisu a *worked* na celkový počet zpracovaných bytů, a řetězce *line* a *end*, do kterých se bude skládat byte po bytu výpis (do *line* začátek řádku a hexadecimální tvar, do *end* ASCII zápis).

Následuje cyklus, který prochází data packetu byte po bytu. V každé iteraci na konci přičte do proměnných *counter* a *worked*.

Pokud je proměnná *counter* rovna 0, implikuje to začátek výpisu na řádek. V tom případě se do proměnné *line* uloží hexadecimálním zápisem počet již zpracovaných (*worked*).

V každé iteraci se zapíše právě zpracovávaný byte v hexadecimální podobě do *line*. Také se ověří, jestli je jeho numerická hodnota menší než 128 a jestli nejde o netisknutelný znak - v takovém případě se do ASCII výpisu (řetězec *end*) uloží daný byte jako znak, v opačném případě je reprezentován znakem `.`.

Proměnná *counter* s hodnotou 7 značí, že se výpis nachází v polovině řádku. V takovém případě se pouze do *line* i *end* vytiskne mezera navíc kvůli formátování. Zajímavější je situace, kdy *counter* dosáhne po přičtení 1 v závěru cyklu hodnoty 16. V ten moment se řetězce *line* a *end* spojí a vytisknout na výstup.

Po skončení cyklu se musí ověřit hodnota proměnné *counter*. Když není 0, zůstaly v řetězcích *line* a *end* zpracované nevytisknuté byty, o které funkce nyní po přidání mezer kvůli konzistenci formátování přidá na výstup.

V celé této části pracuji s *RawCapture* daty paketu. Ze zadání nevyplývá, zda odřádkování v příkladu mělo význam pro oddělení určitých částí dat paketu ani jakých, a proto jsem po analýze výpisu dat paketu ve Wiresharku zvolila stejný způsob: veškerá data paketu zpracovaná v jedné neoddělené části.

3. Testování

Prostředí

Veškeré testování jsem prováděla na referenčním virtuálním stroji. Kromě ověřování správného zacházení s chybnými parametry bylo třeba zkontrolovat správnost výstupních dat. Pro tyto účely jsem využívala nástroj *curl* a *WireShark*.

Curl jsem mimo jiné využila na kontrolu překladu IP adresy na doménové jméno a na vytváření paketů s předvídatelnými vlastnostmi.

WireShark

Tento nástroj byl nápomocný ve více částech tvorby projektu.

Nejprve jsem testovala jeho chování bez ohledu na projekt. Osvěžení znalostí jak vypadá struktura libovolného paketu, jeho zobrazení po bytech ve vztahu k informaci kterou skutečně nese, co se stane na síti po zavolání *ping* na doménu či po použití *curl* z terminálu mi pomohlo ujasnit si co je v projektu zapotřebí.

V největší míře přišel WireShark ke slovu při testování téměř hotového projektu. Pro vyladění chyb stačilo zapnout zachytávání, spustit *ipk-sniffer* a porovnat vypsaný paket s zachyceným.

Příklad porovnání:

Výstup z *ipk-sniffer*:

```
13:43:46.588 162.159.135.234 : 443 > student-vm : 43422

0x0000:  08 00 27 6f 35 b5 52 54  00 12 35 02 08 00 45 00  ..'o5.RT ..5...E.
0x0010:  00 72 3e 0c 00 00 40 06  05 e2 a2 9f 87 ea 0a 00  .r>...@. ........
0x0020:  02 0f 01 bb a9 9e 90 ef  89 d1 eb 13 16 70 50 18  ........ .....pP.
0x0030:  ff ff ed a9 00 00 17 03  03 00 45 b8 2f d5 14 c6  ........ ..E./...
0x0040:  55 65 0e 88 67 53 89 ff  ea 97 d9 03 04 ab f3 f3  Ue..gS.. ........
0x0050:  f9 41 e5 72 bf 8b 72 ac  66 cb 42 c0 6e 66 d3 7f  .A.r..r. f.B.nf..
0x0060:  5b e6 06 fb 5b 86 5a ec  b0 2f 7d 12 0d 96 12 1f  [...[.Z. ./}.....
0x0070:  66 8c d4 21 0c ca 96 c3  84 1a 0b 81 e8 07 55 46  f..!.... ......UF
```

Stejný paket vypsaný pomocí *WireShark*:

```
0000  08 00 27 6f 35 b5 52 54  00 12 35 02 08 00 45 00   ··'o5·RT ··5···E·
0010  00 72 3e 0c 00 00 40 06  05 e2 a2 9f 87 ea 0a 00   ·r>···@· ········
0020  02 0f 01 bb a9 9e 90 ef  89 d1 eb 13 16 70 50 18   ········ ·····pP·
0030  ff ff ed a9 00 00 17 03  03 00 45 b8 2f d5 14 c6   ········ ··E·/···
0040  55 65 0e 88 67 53 89 ff  ea 97 d9 03 04 ab f3 f3   Ue··gS·· ········
0050  f9 41 e5 72 bf 8b 72 ac  66 cb 42 c0 6e 66 d3 7f   ·A·r··r· f·B·nf··
0060  5b e6 06 fb 5b 86 5a ec  b0 2f 7d 12 0d 96 12 1f   [···[·Z· ·/}·····
0070  66 8c d4 21 0c ca 96 c3  84 1a 0b 81 e8 07 55 46   f··!···· ·····UF
```

4. Zdroje

- **Packet Analyser, Wikipedia**

  Wikipedia contributors. Packet analyzer. Wikipedia, The Free Encyclopedia. November 11, 2019, 15:11 UTC. Available at: https://en.wikipedia.org/w/index.php?title=Packet_analyzer&oldid=925665876. Accessed May 3, 2020

- **SharpPCap**

  SharpPCap: A Packet Capture Framework for NET. In: *CodeProject* [online]. 2014 [cit. 2020-05-03]. Dostupné z: https://www.codeproject.com/Articles/12458/SharpPcap-A-Packet-Capture-Framework-for-NET

- **WireShark**

  WireShark [online]. [cit. 2020-05-03]. Dostupné z: https://www.wireshark.org/docs/wsug_html_chunked/