Analýzator komunikačnej siete*

Filip Mojto
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xmojto@stuba.sk

15. október 2023

*Semestrálny projekt v predmete Počítačové a komunikačné siete, ak. rok 2023/24, vedenie: bc. Branislav Jančovič

Obsah

1 Účel dokumentu

Tento dokument slúži ako dokumentácia k 1. projektu z predmetu Počítačové a komunikačné siete. Slúži ako pomôcka pre prácu a skúmanie implementácie komunikačného analýzatora z predmetu PKS. Stručne a v ňom opisujeme funkcionalitu programu, a to aj pomocou diagramov.

2 Základná charakteristika programu

Program je implementovaný sčasti v jazyku C++ and sčasti v jazyku Python. Jeho hlavná úloha spočíva v postupnom spracovaní a následnej analýze skúmaných rámcov z .pcap súborov. Výstupom musia byť analyzované dáta v YAML formáte.

V jazyku c++ je implementovaná samotné čítanie a analyzovanie ramcov. Program pracuje s knižnicou *Npcap*. Analyzátor dokáže na základe prijatých dát o konkrétnom packete zistiť viaceré informácie. Taktiež je možné pomocou prepínača -p filtrovať typy rámcov. Spracované informácie sú následne vložené do textového súboru súboru.

V Pythone bola implementovaná časť na parsovanie získaných dát do formátu YAML. Náš Python skript obsahuje viaceré prepínače, napríklad na určenie cesty k spracovanému textovému súboru alebo prepínač na spracovávanie filtrovaných dát.

Je nutné pripomenúť, že main.cpp súbor sám spúšťa Python skript podľa určenej štruktúry textového i YAML súboru (štruktúra je iná pre spúšťanie programu bez filtrov a s filtrami, i jednotlivé filtre môžu mať inú štruktúru).

Použité knižnice

V tejto časti uvádzame všetky použité knižnice v našej implementácií.

- **npcap** pre prácu analýzu rámcov
- **ruamel** pre formátovanie do YAML v Pythone
- **argparse** pre prácu so vsupnými parametrami

2.1 Rámcový analyzátor - main.cpp

Analyzačná časť projektu obsahuje celkovo 3 zdrojové súbory *main.cpp*, *Network.cpp* a *MyTypes.cpp*. Posledné dva implemetujú každý jeden .h súbor s rovnakým názvom.

Formát vstupného parametra

Súbor *main.cpp* akceptuje nasledovný formát vstupných pametrov:

```
main.cpp -p  PROTOCOL_LABEL
```

V prípade nedodržania potrebného formátu program skončí s chybou.

2.1.1 Funkcia main()

V tejto časti stručne opíšeme implementačnú časť v C++. Budeme sa sústreďovať najmä na dôležité prvky a funkcie.

Program začína metódou *main()*, ktorá príjima vstupné parametre. V našome prípade to bude použitie prepínača -p v kombinácii s niektorým podporovaným protokolom.

Následne sa načítajú podporované protokoly z jednotlivých súborov do patričných statických polí.

```
1965  int main(int argc, char* argv[]) {
1966
1967      //Firstly, lets read all necessary data from external files to the respective arrays (etherTypes, lsaps)
1968      load_protocols(L_3_FILE_PATH, l_3_protocols);
1969      load_protocols(L_2_FILE_PATH, l_2_protocols);
1970      load_protocols(L_1_FILE_PATH, l_1_protocols);
1971      load_protocols(LSAPS_FILE_PATH, lsaps);
1972      load_protocols(ICMP_TYPES_FILE_PATH, icmp_types);
1973
1974      //We process the input parameters here and create ProtocolFilter struct accordingly
1975      if (process_param(argc, argv) == -1) {
1976          return -1;
1977      }
1978
1979      //Now, lets start launching a session for the input file
1980      char source[PCAP_BUF_SIZE];
1981      pcap_t* handle;
1982
1983      if (get_src_str(source) != 0 || get_session_handle(handle, source) != 0) {
1984          return -1;
1985      }
1986
1987      //After achieving the previous steps, we can now insert a header into the input file for a python script
1988      fileStream.open(SCRIPT_INPUT_FILE_PATH);
1989      insert_header_data();
1990
1991      //Now, the main process of processing and analyzing packets begins
1992      pcap_loop(handle, 0, handlePacket, NULL);
1993      call_parser_script();
1994
1995      ////At the end, we close and dealloc some variables
1996      fileStream.close();
1997      pcap_close(handle);
1998      delete protocol_filter;
1999
2000      return 0;
2001  }
```

Obr. 1: main.cpp

V druhom kroku sa spracujú prípadné vstupné parametre, overí sa ich platnosť a štruktúra. Na základe nich je vytvorená štruktúra *ProtocolFilter*, ktorá obsahuje okrem iného názov zvoleného protokolu. Nakoľko je vytvorená globálne, môžeme s ňou pracovať vo funkcii *handlePacket()*.

Nasleduje nakonfigurovanie spracovávania rámcov pomocou knižnice *Pcap* a otvorenie spojenia. Takisto sa nadviaže aj súborový prúd do textového súboru, do ktorého sa okamžite pomocou funkcie *insert_header_data()* vloží hlavička.

Program teraz prejde na hlavný proces spracovávania a analýzy prijatých paketov pomocou funkcie *pcap_loop()*, ktorá pri každom spracovanom packete zavolá funkciu handlePacket(). Túto funkciu si popíšeme v jednej z nasledujúcich častí dokumentu.

Na záver sa zavolá Python skript, ktorý spracuje údaje o jednotlivých paketoch v súbore *input.txt*.

2.1.2 Ďalšie dôležité štruktúry a funkcie

Začnime vypísaním dôležitých štruktúr, ktoré sú združené vo vektoroch tak, aby vytvárali jednotlivé komunikácie.

1. TCP_Comm
2. ARP_Base
3. ICMP_Comm
4. UDP_Comm
5. Request
6. ICMP_Request

7. UDP_Request

Prvé 4 reprezentujú buď samotné komunikácie alebo, ako v prípade ARP, slúžia na združovanie komunikácií. Request štruktúra a jej varianty pre ďalšie protokoly slúži ako samotný rámec, je to teda jeden element v komunikácii. Tu uvádzame zoznam niektorých ďalších dôležitých funkcií. Tieto zvyčajne používajú ďalšie svoje pomoconé funkcie a procedúry.

1. *handlePacket()*
2. *process_tcp_req()*
3. *process_arp_req()*
4. *process_icmp_req()*
5. *process_tftp_req()*
6. *process_comms()*

Prvá funkcia je volaná vždy, keď sa príjme nový rámec. Jej hlavnou úlohou je naplniť objekt triedy *MyTypes::Dictionary* dátami získanými z postupnosti bytov a tú potom poslať na spracovanie podľa zisteného protokola niektorej z funkcií 2 - 5. Spomínaná trieda obsahuje nami implementovanú funkciu na načítavanie dvojíc kľúč - hodnota. Okrem iného funkcia aj zahŕňa filtrovanie protokolov tak, že daný rámec pošle na spracovanie iba vtedy, ak jeho protokol pochádza z rovnakej alebo vyššej vrstvy ako protokol na vstupe.

Ďalšie 4 funkcie sa volajú podľa prijatého protokolu a obsahujú komplexnú funkcionalitu na spájanie rámcov do komunikácií. Ich hlavnou zodpovednosťou je umiestniť dané rámce do ich štruktúr tak, aby boli požadované rôzne požiadavky v súvislosti s komunikáciami.

Posledná funkcia je zodpovedná zato, aby vypísala dáta uchované v poliach a vektoroch do textového súboru v požadovanom formáte.

2.2 Formátovanie na YAML - packet_parser.py

Táto časť má za úlohu prijať textový súbor, ktorý je výstupom analyzačnej časti a preformátovať ho na YAML formát do súboru *output.yaml*.

2.2.1 Štruktúra vstupných parametrov

Skript používa na parsing vstupných parametrov knižnicu *argparse*. V skripte máme definovaných niekoľko možných parametrov:

1. -i
2. -o
3. -a
4. -p

Prvé dve z nich slúžia na konfiguráciu vstupných a výstupných súborov. Parameter *-a* nám umožňuje vpisovať údaje o rámcoch nakoniec výstupného súboru namiesto toho, aby ho predtým kompletne vymazal. Parameter -p je kľúčový a slúži na prepnutie YAML štruktúry na základe vyfiltrovaného protokolu. Pre prípadné ďalšie informácie možno použiť klasicky prepínač *-h*.

2.2.2 Funkcia main

Funkcia najskôr nakonfiguruje podľa potreby náš mechanizmus príjimania vstupných parametrov. Následne sa vyčistia všetky potenciálne dáta z výstupného súboru, ak je prítomný prepínač *-a*.

Ďalším krokom je načítanie dát z výstupného súboru do premennej, do ktorej sa následne podľa načíta hlavička. Tá je pomocou funkcie *parse_header_data()* naplnená dátami so vstupného súboru.

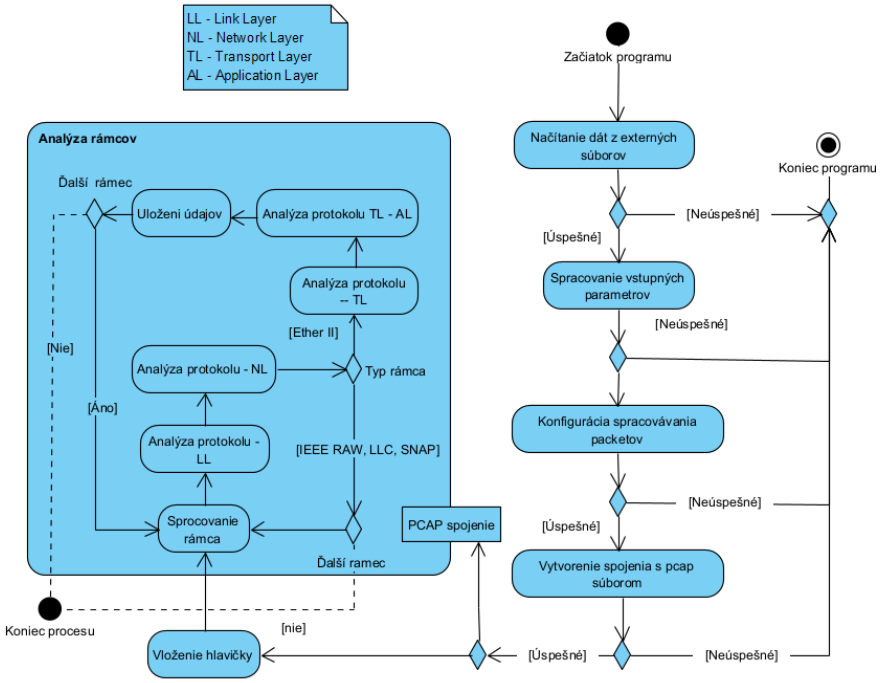
Nasleduje hlavný proces preformátovania textového súboru na YAML pomocou funkcie *process_input()*. Táto funkcia spracuje podľa rôznych dohodnutých delimitrov v textovom súbore dané dáta a prepisuje ich postupne do výstupného súboru.

```
if __name__ == "__main__":
    configure_argparse()

    if not args.append:
        clear_data()

    data = load_data(yaml)
    data = set_header(data)

    #Managing header in case of '-a' parameter
    if not args.append:
        parse_header_data()
        file_input = skip_header()

    file_input = process_input(file_input)

    save_data(data, yaml)
```

Obr. 2: main() funkcia v packet_parser.py

3 Modelovanie implementácie

V tejto časti uvádzame našu implementáciu analýzy rámcov vo forme diagramu aktivít. Ten zachýtava najdôležitejšie časti spracovávania rámcov, od načítania podporovaných protokolov až po úplné ukončenie analýzy.



Obr. 3: Model procesu analýzy rámcov

Samotná analýza rámcov je v diagrame zobrazená ako aktivita, ktorá obsahuje niekoľko akcií. Tieto akcie predstavujú postupné spracovávania skúmaného rámca po jednotlivých vrstvách.

Pre prehľadnosť sme v diagrame neuvádzali funkciu obmedzenia analýzy len na určité protokoly (filtre). Tie fungujú tak, že postupne od linkovej vrstvy sa kontroluje typ protokolu a ak sedí, tak je spracovanie akceptované pre akýkoľvek prípadný protokol nachádzajúci sa na vyššej vrstve.

4 Zhodotenie a záver

Úlohou bolo implementovať komplexný rámcový analýzátor komunikačnej sieti. Musíme povedať, že napriek mnohým problémom a prekážkam sa nám tento cieľ podarilo naplniť vcelku uspokojivo. Najväčším problémom bola implementácia analýzy TCP rámcov, ktoré majú komplexnú logiku komunikácie.

K dispozícii sme mali aj vzorové .pcap súbory, na ktorých sme si mohli otestovať naše riešenie. Vo veľa prípadoch sme zistili, že naša implementácia má stále svoje nedostatky a nezachýtáva rôzne prípady, ktoré mohli v komunikáciach nastať.

Okrem dôvodu splnenia podmienok pre semestálné zadanie sme tento projekt využili aj na rozšírenie našich vlastných vedomostí v danej oblasti.