Academia Tehnică Militară “Ferdinand I”
Proiect – Proiectarea Sistemelor de Operare

# APLICAȚIE

# TIP PORT SCANNER


Îndrumător:
Slt. Adina VAMAN

Realizat de:
Sd. Sg. Sorin-Ionuț MIHALI
Sd. Sg. Maria-Emilia GRIGORE
Grupa C113C

# Cuprins

# Capitolul 1 – Introducere

## 1.1 Scopul proiectului

Crearea unui produs software care să permită utilizatorului scanarea porturilor de rețea, pentru vizualizarea traficului(transmisie-recepție), dar și a stadiului în care acestea se află(open/not open).

## 1.2 Lista definițiilor

Scanarea porturilor dintr-o rețea este o metodă prin care putem să determinăm care porturi sunt deschise și ar putea permite primirea/trimiterea de pachete în rețeaua respectivă. Această metodă presupune, de asemenea, și trimiterea de pachete pe anumite porturi pentru a vedea răspunsurile primite și a identifica, astfel, vulnerabilitățile existente (exemplu: acces neautorizat). Protocoalele folosite pentru scanarea de porturi sunt TCP si UDP.

Porturi hardware: mufe pentru periferice

Porturi software: un segment/parte/bucată de cod software căreia i-a fost asignat un alt sistem de operare sau componentă hardware pe care să lucreze față de cel/cea original/ă.

Porturi de rețea: valoare numerică asociată cu un protocol care facilitează comunicarea pentru un serviciu/funcție

### 1.3 Structura documentului

Documentul este împărțit în cinci capitole. Capitolul 1 reprezintă introducerea. Capitolul 2 prezinta structura aplicației, fișierele încorporate și entitățile din cadrul lor. Capitolul 3 modul și principiile de funcționate din spatele aplicației . Capitolul 4 cuprinde exemple de testare a funcționalităților aplicației. Capitolul 5 prezintă concluziile în raport cu structura inițială a aplicației.

# Capitolul 2 – Structura Aplicației

Programul conține:

- Fișierul ***arg_parse.h***:
  Cuprinde structura arguments care are ca membri toate opțiunile pe care le are la dispoziție utilizatorul. De asemenea, aici este implementată și funcția *error_t parse_opt(int key, char *arg, struct argp_state *state)* cu ajutorul căreia se recunosc opțiunile și parametrii transmiși de către utilizator.

```c
struct arguments
{
    char host[INET_ADDRSTRLEN]; // hostname sau IP
    int timeout;                // timeout pentru fiecare port
    int no_threads;                // numar de thread-uri
    char file_to_output[30];    // fisier de output
    char file_to_input[30];     // fisier de input
    int start_port;             // port inceput range
    int end_port;               // port sfarsit range
    char scan_type[10];         // tipul scanarii: TCP sau UDP
    int verbose;                // verbose
    int randomize;              // scanarea porturilor in ordine aleatoare
    int fast;                   // scanare rapida
    int *excluded_ports;        // range de porturi excluse de la scanare
    int excluded_ports_count;
    int tcp_flags[7]; // TCP flags la scanare
    int flag;         //
};
```

```c
error_t parse_opt(int key, char *arg, struct argp_state *state)
{
    struct arguments *arguments = (struct arguments *)state->input;

    switch (key)
    {
    case 'h':
        strncpy(arguments->host, arg, (size_t)INET_ADDRSTRLEN);
        break;
    case 't':
        arguments->timeout = atoi(arg);
        break;
    case 'o':
        strncpy(arguments->file_to_output, arg, 30);
        break;
    case 'i':
        strncpy(arguments->file_to_input, arg, 30);
        break;
```

- Fișierul ***myScan.c:***
  Cuprinde funcția main în cadrul căreia se realizează procedeul de scanare prin apelarea mai multor funcții: *input_file_parse(FILE *f, arguments *args), void set_tcp_flags(int *flags, int *flag), void create_thread(struct arguments user_args), void *scan_thread(void *arg)*, etc.

```c
/*Parsam argumentele programului, cu ajutorul header-ului argv_parse.h
si interpretam argumentele date*/
int main(int argc, char *argv[])
{
    struct arguments user_args;
    user_args = parse_args(argc, argv);

    struct hostent *target;
    int rc, fd;

    /*daca optiunea -o e setata = scriem outputul intr-un fisier*/
    if (strlen(user_args.file_to_output) != 0)
    {
        test_output_file(&fd, &rc, &(user_args.file_to_output));
    }

    /*setare scan-type*/
    if (strcmp(user_args.scan_type, "TCP") == 0)
    {
        /*daca e scanare de tip tcp*/
        set_tcp_flags(&(user_args.tcp_flags), &(user_args.flag));
    }
```

```c
void create_thread(struct arguments user_args)
{
    int thread_id, check = 0;
    pthread_t threads[user_args.no_threads];
    struct thread_options opt[user_args.no_threads];

    /*Creare thread-uri*/
    for (thread_id = 0; thread_id < user_args.no_threads; thread_id++)
    {
        opt[thread_id].thread_id = thread_id;
        opt[thread_id].excluded_ports = user_args.excluded_ports;
        opt[thread_id].excluded_ports_count = user_args.excluded_ports_count;
        opt[thread_id].fast = user_args.fast;
        strcpy(opt[thread_id].host, user_args.host);
        opt[thread_id].randomize = user_args.randomize;
        opt[thread_id].timeout = user_args.timeout;
        strcpy(opt[thread_id].scan_type, user_args.scan_type);
        opt[thread_id].verbose = user_args.verbose;
        strcpy(opt[thread_id].file_to_output, user_args.file_to_output);
        opt[thread_id].flag = user_args.flag;

        for (int i = 0; i < 7; i++)
            opt[thread_id].tcp_flags[i] = user_args.tcp_flags[i];
```

- Fișierul ***myScan.h***:
  Conține structura *thread_options* corespunzătoare unui thread.

```c
struct thread_options {
    char host[INET_ADDRSTRLEN]; //inet_addrstrlen = 16
    int port;
    pthread_t thread_id;
    int timeout;                // timeout pentru fiecare port (cate secunde sa astepte pana raspunde un port)
    int threads;                // numar de thread-uri
    char file_to_output[30];    // fisier de output
    char file_to_input[30];     // fisier de input
    int start;                  // port inceput range
    int end;                    // port sfarsit range
    char scan_type[10];         // tipul scanarii: TCP sau UDP
    int verbose;                // verbose
    int randomize;              // scanarea porturilor in ordine aleatoare
    int fast;                   // scanare rapida
    int *excluded_ports;        // range de porturi excluse de la scanare
    int excluded_ports_count;   // numarul de porturi excluse
    char tcp_flags[7];          // TCP flags la scanare
    int flag;
};
```

- Fișierul Makefile

Makefile

```
all: myScan

myScan: myScan.c
    gcc -pthread -w -o myScan myScan.c
```

# Capitolul 3 – Mod de funcționare

Utilizatorul rulează aplicația din terminal ca orice alt executabil, cu comanda *./myScan*. Acesta poate introduce diverse opțiuni, după cum urmează:

- **hostname sau ip [-h]:** numele de domeniu sau adresa ip a serverului căruia dorim să-i scanăm porturile;
- **input file [-i]:** ia adresele ip sau numele de domeniu dintr-un fișier dat ca argument;
- **timeout [-t]:** specifică timpul maxim (în secunde) pentru așteptarea ca un port să răspundă;
- **output file [-o]:** scrie outputul comenzii într-un fișier;
- **port range [-p]:** scanează:
  - -un port (80);
  - -un range de porturi (75-85);
- **excluded ports [-e]:** în scanare, sare peste:
  - -un range de porturi (45-90);
  - -o mulțime de porturi (45,89,1234);
  - -un singur port: (89);
- **threads [-T]:** setează un alt număr de threaduri folosite în program față de cel implicit (adică 5);
- **verbose [-v]:** specifică și tipul de serviciu găsit pe port și tipul de protocol;
- **random [-r]:** scanează porturile într-o ordine aleatoare;
- **fast [-f]:** nu se introduce o întârziere în scanare;
- **scanType [-s]:** specificăm tipul de scanare: TCP sau UDP;
- **tcp-flags [-F]:** specificăm flagurile din headerul TCP: [S] = SYN, [F] = FIN, [FPU] = XMAS.

După parsarea argumentelor, în funcția main se verifică dacă utilizatorul a introdus un hostname, moment în care se translatează în adresă IP prin funcția *gethostbyname()*. Tot în main se verifică dacă este activată opțiunea -i (input file) pentru a citi adresele destinație din fișier sau opțiunea -o (output file) pentru a transmite output-ul comenzii într-un fișier dat ca argument. De asemenea este verificată opțiunea scanType. Pentru TCP, sunt setate flagurile prin funcția *set_tcp_flags()*.

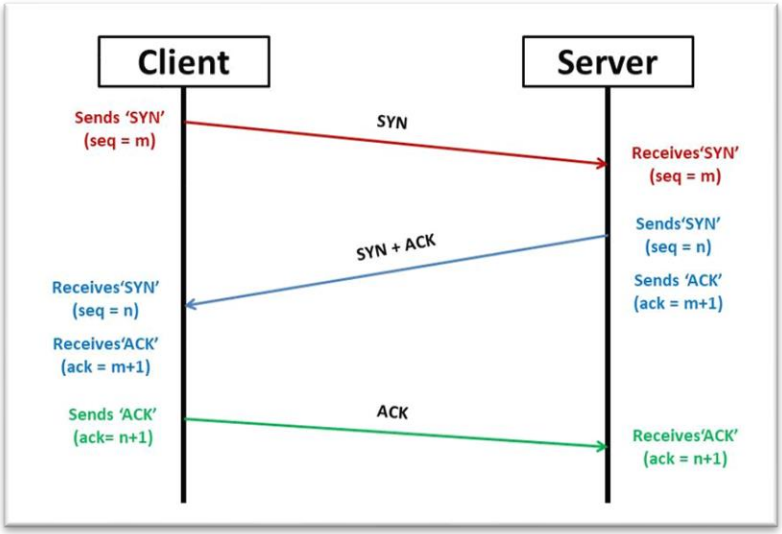
Începe crearea threadurilor prin apelul funcției *create_thread()*. Aici se populează structura de tip *thread_options* cu argumentele corespunzătoare. Se

apelează *pthread_create()* cu rutina de execuție a threadului specificată în funcția *scan_thread()*.

În funcția *scan_thread()* se verifică opțiunea -r (randomize) pentru a stabili dacă threadul va scana un port la întâmplare sau va urma ordinea crescătoare, de la 0 la 1024 (sau alt interval specificat de utilizator cu opțiunea -p). De asemenea, se ține cont și de opțiunile -f (fast = așteptare redusă pentru răspunsul unui port) și -e (excluded ports = porturi care nu se verifică).

Se urmează două cazuri:

- **Scanare TCP**, cu alte 4 cazuri aferente celor 4 flaguri care pot fi setate pentru acest tip de scanare:
  - **TCPConnect:** Se creează un obiect de tip socket care folosește protocolul TCP (IPPROTO_TCP), care se populează cu argumentele corespunzătoare prin funcția *setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (const char *)&args->timeout, sizeof(tv));* Se încearcă conexiunea la port prin funcția connect().
  - **SYN, FIN, XMAS:** Se creează un obiect de tip socket raw care folosește tot protocolul TCP (IPPROTO_TCP). SYN, FIN si XMAS funcționează pe principiul Three Way Handshake:




Este aflată prin funcția *get_local_ip()* adresa IP a calculatorului sursă (de pe care ruleaza aplicația). Sunt declarate și 2 obiecte de *iphdr* și *tcphdr* pentru a construi un pachet care poate fi transmis socketului destinație,

pentru verificarea conexiunii cu portul. Starea portului poate fi OPEN, CLOSED sau UNREACHABLE (cele de acest tip nu sunt afișate). După ce se încearcă transmiterea pachetului, se așteaptă un răspuns prin funcția *recvfrom()*. Dacă funcția returnează un număr mai mic decât 0, înseamnă că nu a reușit încercarea de conectare cu portul. Pentru fiecare dintre cele 3 flag-uri se afișează mesaje corespunzătoare în caz de eroare sau, în caz de conexiune reușită, statusul portului găsit prin funcția *display_port_status()*.

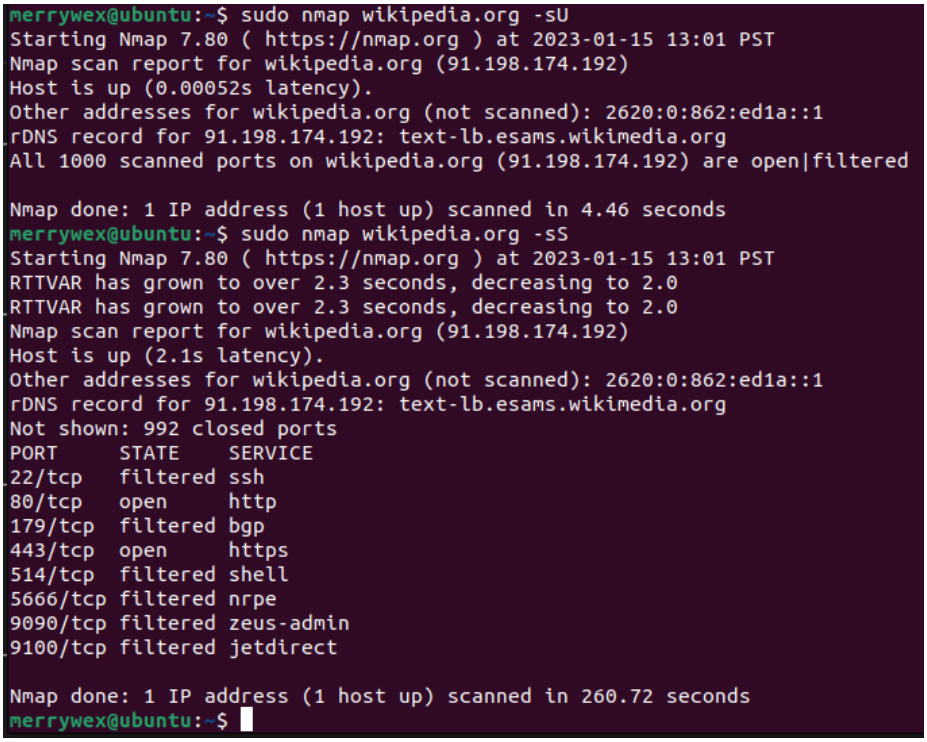
- **Scanare UDP:** Se creează un socket care folosește protocolul UDP (IPPROTO_UDP). Se urmează același parcurs de încercare de conectare la portul destinație. În caz de reușită, se afișează portul.

Se ține cont de opțiunea -v (verbose) care afișează în plus și serviciul care rulează pe portul respectiv și protocolul folosit.

În mod implicit, programul va rula cu următoarele argumente:
-host=””, programul nu va rula fără acest argument;
-input file=””;
-output file=””;
-timeout=3;
-threads=5;
-scan_type=”TCP”;
-start_port=1;
-end_port=1024;
-excluded_ports=NULL;
-excluded_ports_count=0;
-tcp_flags=0;
-flag=0;
-randomize=0;
-fast=0;
-verbose=0;

# Capitolul 4 – Testare funcționalități

Verificările se vor face pentru numele de domeniu wikipedia.org. Ca referință, outputul comenzii *nmap wikipedia.org* este:

**Test 1- Verificare opțiune -h cu un nume de domeniu:**

```
merrywex@ubuntu:~/psop$ sudo ./myScan -h wikipedia.org
Scanning with TCPConnect.
Scanning 91.198.174.192
--> Created 5 threads.
PORT: 443       STARE: OPEN
PORT: 80        STARE: OPEN
```

**Test 2- Verificare opțiuni -i, -o și -p:**

```
merrywex@ubuntu:~/psop$ ./myScan -p 75-85 -i in.txt -o out.txt
```

in.txt

```
1   mta.ro
2   google.com
3   8.8.8.8
4   127.0.0.1
```

out.txt

```
1   Scanning with TCPConnect.
2   Scanning 192.124.249.79
3   --> Created 5 threads.
4   PORT: 80    STARE: OPEN
5   Scanning 142.250.180.206
6   --> Created 5 threads.
7   PORT: 80    STARE: OPEN
8   Scanning 8.8.8.8
9   --> Created 5 threads.
10  PORT: 80    STARE: OPEN
11  Scanning 127.0.0.1
12  --> Created 5 threads.
13  PORT: 80    STARE: OPEN
14
```

**Test 3- Verificare opțiune -v:**

```
merrywex@ubuntu:~/psop$ ./myScan -h wikipedia.org -p 75-85 -v 1
Scanning with TCPConnect.
Scanning 91.198.174.192
--> Created 5 threads.
PORT: 80        STARE: OPEN SERVICE:http        PROTOCOL:tcp
```

**Test 4- Verificare opțiuni -s TCP și -F SYN:**

```
merrywex@ubuntu:~/psop$ sudo ./myScan -h wikipedia.org -p 80 -s TCP -F S
Scanning with SYN.
Scanning 91.198.174.192
--> Created 1 threads.
Packet sent.
PORT: 80        STARE: OPEN
```

**Test 5- Verificare opțiuni -s TCP și -F FIN:**

```
merrywex@ubuntu:~/psop$ sudo ./myScan -h 8.8.8.8 -s TCP -F F -p20-25
Scanning with FIN.
Scanning 8.8.8.8
--> Created 5 threads.
Packet sent.
Packet sent.
Packet sent.
Packet sent.
Packet sent.
PORT: 23        STARE: OPEN
PORT: 24        STARE: OPEN
PORT: 22        STARE: OPEN
PORT: 21        STARE: OPEN
PORT: 25        STARE: OPEN
```

**Test 6- Verificare opțiuni -s TCP și -F FPU(XMAS)**

```
ionut@ionut-virtual-machine:~/PSOP$ sudo ./myScan -h 8.8.8.8 -s TCP -F FPU -p 4443 -v 1 -f 1
Scanning with XMAS.
Scanning 8.8.8.8
--> Created 1 threads.
Packet sent.
PORT: 4443      STARE: OPEN
```

### Test 7- Verificare opțiuni -s UDP și -T:

```
merrywex@ubuntu:~/psop$ sudo ./myScan -h wikipedia.org -s UDP -T 100
Scanning with UDP.
Scanning 91.198.174.192
--> Created 100 threads.
PORT: 2 STARE: OPEN
0
PORT: 32        STARE: OPEN
1
PORT: 42        STARE: OPEN
2
PORT: 52        STARE: OPEN
3
PORT: 62        STARE: OPEN
4
PORT: 182       STARE: OPEN
5
PORT: 172       STARE: OPEN
6
PORT: 192       STARE: OPEN
7
PORT: 202       STARE: OPEN
8
PORT: 212       STARE: OPEN
9
PORT: 222       STARE: OPEN
10
PORT: 22        STARE: OPEN
11
PORT: 232       STARE: OPEN
12
```

**…nu incap toate in screenshot. Sunt 1000.**

### Test 8- Verificare opțiune -e:

- Output fără opțiunea -e:

```
ionut@ionut-virtual-machine:~/PSOP$ ./myScan -h wikipedia.org -p75-85 -f 1
Scanning with TCPConnect.
Scanning 91.198.174.192
--> Created 5 threads.
PORT: 80        STARE: OPEN
```

- Output cu opțiunea -e:

```
ionut@ionut-virtual-machine:~/PSOP$ ./myScan -h wikipedia.org -p75-85 -f 1 -e80
Scanning with TCPConnect.
Scanning 91.198.174.192
--> Created 5 threads.
```

## Test 9- Verificare opțiune -r:

- Output fără opțiunea -r:

```
ionut@ionut-virtual-machine:~/PSOP$ sudo ./myScan -h wikipedia.org -p20-25 -f 1 -s UDP -T 1
Scanning with UDP.
Scanning 91.198.174.192
--> Created 1 threads.
PORT: 21        STARE: OPEN
PORT: 22        STARE: OPEN
PORT: 23        STARE: OPEN
PORT: 24        STARE: OPEN
PORT: 25        STARE: OPEN
```

- Output cu opțiunea -r:

```
ionut@ionut-virtual-machine:~/PSOP$ sudo ./myScan -h wikipedia.org -p20-25 -f 1 -s UDP -T 1 -r 1
Scanning with UDP.
Scanning 91.198.174.192
--> Created 1 threads.
PORT: 24        STARE: OPEN
PORT: 22        STARE: OPEN
PORT: 23        STARE: OPEN
PORT: 21        STARE: OPEN
```

# Capitolul 5 – Concluzii

Inițial, aplicația era concepută așa fel încât să nu acopere prea multe funcționalități, așa cum se poate observa din livrabilul 1:

*Fereastra ”Scan Window”:*
*Implicit, aplicația va scana toate porturile de la 1 la 1024.*

- *Opțiunea ”Ports”: scanarea se va efectua pe un port/listă de porturi.*
- *Opțiunea ”IP”: permite utilizatorului să scaneze o adresă IP individuală.*
- *Opțiunea ”File”: se dă ca argument un nume de fișier care conține pe fiecare linie câte o adresă IP, pe care se va face scanarea.*
  **nota: în cazul în care nu e specificată opțiunea ”IP” și opțiunea ”File” sau dacă fișierul dat ca argument la opțiunea ”File” este gol, va apărea o eroare și utilizatorul va avea posibilitatea de a completa parametrii din nou.*
- *Opțiunea ”Transport”: implicit, se scanează atât porturile UDP, cât și TCP. Utilizând această opțiune, putem alege să facem scanarea doar pe unul dintre aceste tipuri.*
- *Opțiunea ”Help”: oferă utilizatorului un man-page.*

Am implementat 8 funcționalități în plus față de cele gândite inițial (output file [-o], excluded ports [-e], timeout [-t], threads [-T], verbose [-v], random [-r], fast [-f], TCP flags [-F]).

De asemenea, s-ar fi dorit crearea unei interfețe grafice pentru aplicație, dar am considerat că este în regulă și simpla rulare în linie de comandă.

Față de varianta 1 a aplicației, am considerat necesară implementarea multithreading, întrucât scanarea dura foarte mult timp. Astfel, fiecare thread se ocupă de scanarea unui range de porturi proporțional distribuite, iar execuția se încheie semnificativ mai rapid.