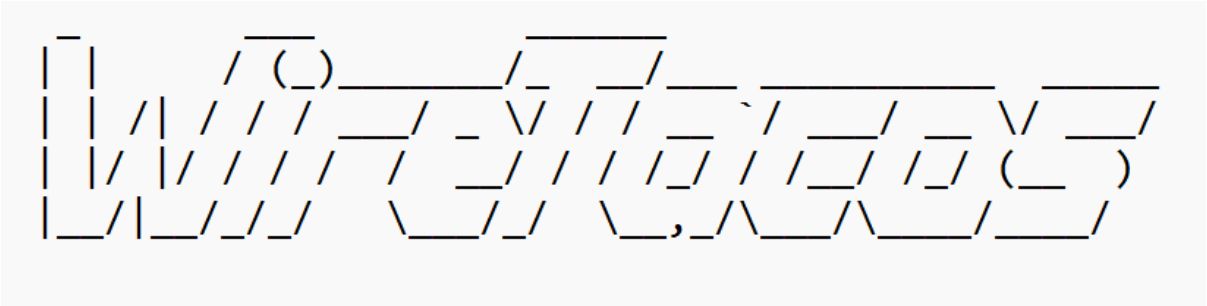
READ ME



Étudiants: Audigier Roman
Iordache Paul-Tiberiu

Groupe: 8

Année universitaire: 2022-2023

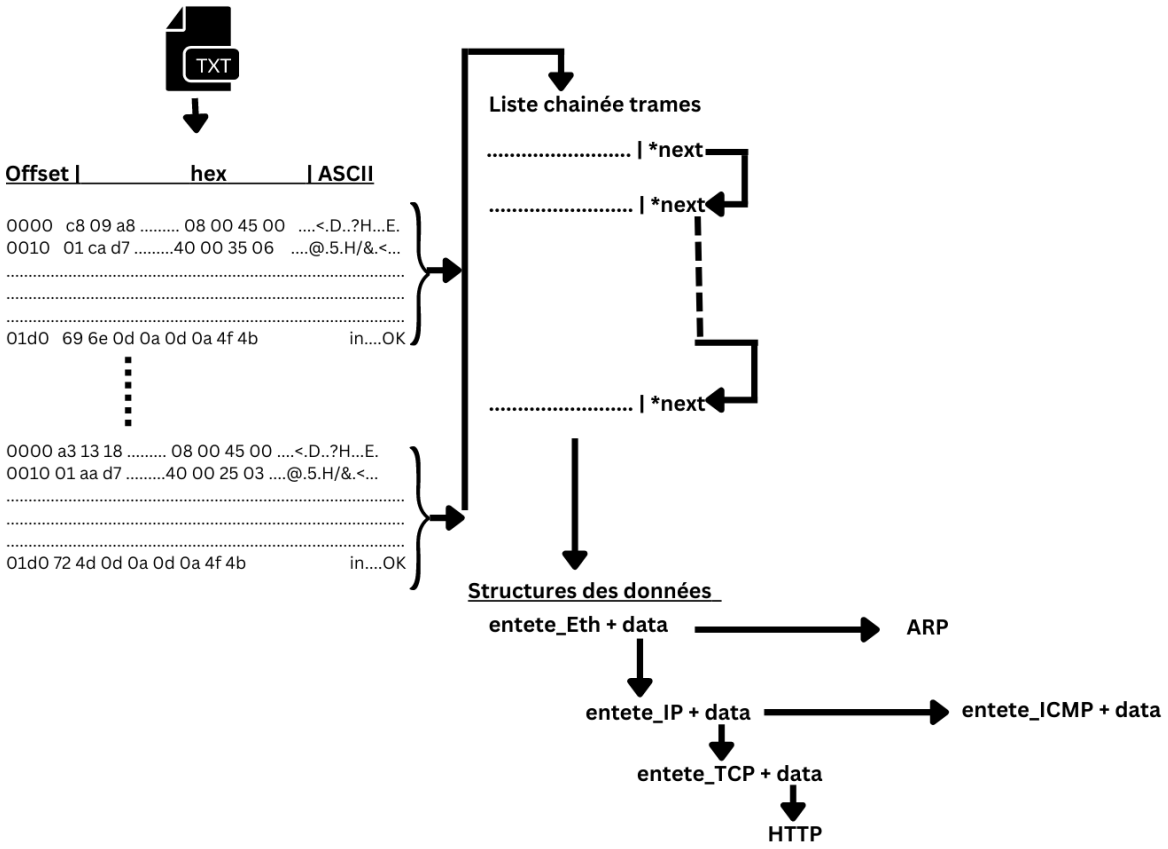
Sommaire

I. Introduction au logiciel WireTacos

Le logiciel WireTacos n'est pas un analyseur de trames, mais un visualiseur de trafic, bien qu'il pourrait l'être également. En effet, la philosophie derrière notre logiciel a été de créer une base de code capable d'analyser toutes les informations présentes dans une trame en format Hex Dump afin d'en faire l'affichage voulu, ici, il cherche à répliquer la fonctionnalité "Flow Graph" de WireShark mais en CLI. L'approche quasi-OOP, avec une structure par entête de protocole, permet une implémentation facile de nouveaux protocoles. Le choix du langage C nous permet d'avoir une base robuste provoquant peu d'erreur à l'exécution plus rapide qu'un code équivalent en Python ou bien en Java.

II. Présentation de la structure du projet

Le choix de séparer notre code en différents fichiers s'inscrit à la fois dans la démarche de vouloir un code modulaire, où l'ajout d'un nouveau protocole à analyser se fait dans un nouveau fichier .c, mais aussi pour augmenter la lisibilité de notre projet. Pour mieux comprendre les structures des données utilisées dans notre projet, nous avons fait un schéma qui explique comment nous avons stocké en mémoire les trames. Après avoir lu le fichier .txt, nous utilisons des listes chaînées (de chaînes de caractères) pour stocker chaque trame. Puis, pour arriver à l'affichage, on utilise les fonctions spécifiques pour chaque entête et protocole que notre visualisateur est capable de traiter.

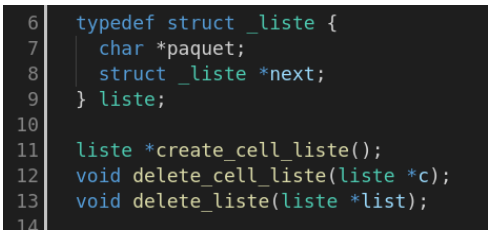


A. Fonctions utiles pour le format de trames

Dans les fichiers format.c et format.h, nous avons 5 fonctions qui sont utiles pour le formatage des adresses ainsi que pour leur affichage.

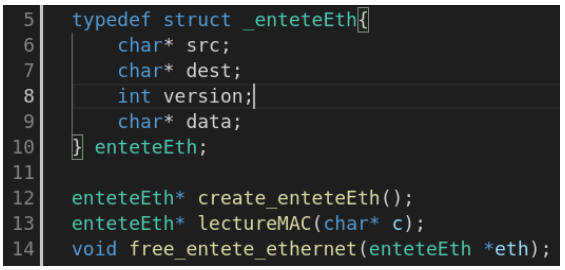
```
4  char* space_erase(char*src);
5  char* MACformat(char*src);
6  char* IPformat(char*src);
7  int hex_to_int(char*string);
8  char hex_to_char(char* string);
9  int stringcompare(char* s1, char* s2);
```

B. Fonctions pour manipuler les listes de trames



Dans les fichiers liste.c et liste.h, nous avons des fonctions classiques qui nous aident pour la manipulation de listes. A l'aide de ces listes, nous avons pu stocker chaque paquet contenu dans le fichier donné. Après, avec l'aide de la structure de liste chaînée, on peut afficher le contenu de chaque paquet.

C. ethernet.c et ethernet.h



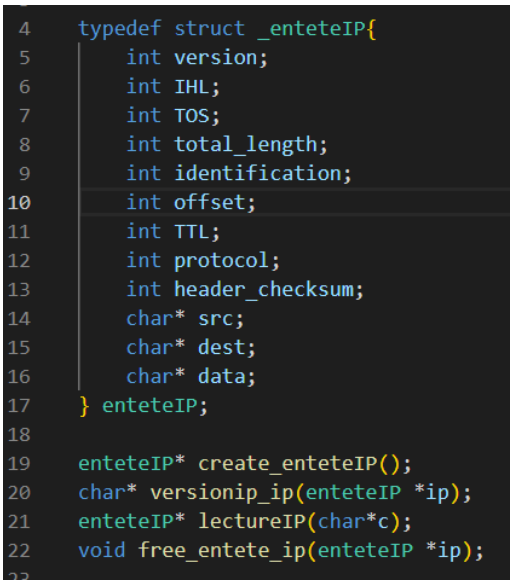
Dans les fichiers ethernet.c et ethernet.h nous avons 3 fonctions qui nous aide pour le traitement de l'entête ethernet. La fonction la plus importante est: enteteEth*lectureMAC(char* c), qui va nous aider à stocker en mémoire les données importantes trouvées dans l'entête ethernet, ainsi que ses données. L'entête ethernet étant fixe, l'utilisation de sscanf avec le format %17[0-9a-fA-F] pour les adresses MAC, %5[0-9a-fA-F] pour la version et %[0-9a-fA-F] pour data, est la ligne de code qui nous a permis de stocker les informations de l'entête. En plus, les fonctions MACformat et hex_to_int ont aidé pour stocker les chaînes de caractères dans le format souhaité.

D. ARP.c et ARP.h

Dans ces 2 fichiers, nous avons implémenté des fonctions qui nous aident pour le traitement du protocole ARP, qui est encapsulé dans la section data de ethernet. Comme pour ethernet, ARP a un entête de taille fixe donc on a toujours utilisé sscanf. En plus, l'utilisation des

fonctions IPformat et MACformat, ont aidé pour stocker les chaînes de caractères dans le format souhaité.

E. IP.c et IP.h

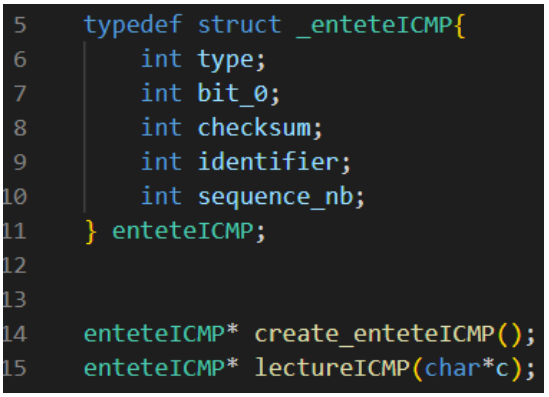


```
typedef struct _enteteIP{
    int version;
    int IHL;
    int TOS;
    int total_length;
    int identification;
    int offset;
    int TTL;
    int protocol;
    int header_checksum;
    char* src;
    char* dest;
    char* data;
} enteteIP;

enteteIP* create_enteteIP();
char* versionip_ip(enteteIP *ip);
enteteIP* lectureIP(char*c);
void free_entete_ip(enteteIP *ip);
```

En ce qui concerne IP, on utilise toujours deux fichiers. Notre analyseur sera capable d'afficher seulement les trames IPv4. Dans ce cas, on peut déterminer la taille de l'entête à l'aide du champ IHL (pour savoir si on a des options). Dans la fonction lectureIP, on a toujours utilisé sscanf (comme pour ethernet et ARP) pour lire les informations qu'on va stocker en mémoire (à l'aide des fonctions de formatage space_erase et IPformat). En plus, on a utilisé une chaîne de caractères (char*optionlessdata) ou on va stocker toutes les options IP. Puis, si la taille de notre entête est supérieure à 20 octets. Si c'est le cas, alors on va stocker les données des options dans la section data de la structure IP.

F. ICMP.c et ICMP.h



```
typedef struct _enteteICMP{
    int type;
    int bit_0;
    int checksum;
    int identifier;
    int sequence_nb;
} enteteICMP;


enteteICMP* create_enteteICMP();
enteteICMP* lectureICMP(char*c);
```

ICMP est un protocole qui se trouve en IP, donc il va être stocké dans la section data de IP. Comme avant, la fonction de lecture va être la plus importante. Nous utilisons toujours sscanf pour lire les informations qu'on veut stocker (bien sûr avec l'aide de la fonction de formatage hex_to_int). En plus, a l'aide de ce protocole, on va déterminer s'il s'agit d'un echo request ou d'un echo reply.

G. TCP.c et TCP.h

```
5   typedef struct _enteteTCP{
6       int port_src;
7       int port_dst;
8       unsigned long sequence_nb;
9       unsigned long ack_nb;
10      int THL;
11      char* reserved_flags;
12      int window;
13      int checksum;
14      int urgent_pointer;
15      char* data;
16  } enteteTCP;
17
18  enteteTCP* create_enteteTCP();
19  enteteTCP* lectureTCP(char* c);
20  void free_enteteTCP(enteteTCP *tcp);
```

Comme d'habitude, pour TCP (encapsulé dans un paquet IP), la fonction la plus importante va être lectureTCP. Cette fois ci, pour stocker les chaînes de caractères, nous avons choisi d'utiliser la fonction hex_to_int, puis strtol pour pouvoir stocker les champs sequence number et acknowledgment number (codés sur 32 bits, donc on a choisi de le stocker en unsigned long). Pour lire les informations qu'on veut stocker, on utilise toujours sscanf.

H. HTTP.c et HTTP.h

```
9   typedef struct _HTTP{
10      char* ligne;
11      char* data;
12  } HTTP;
13
14
15  HTTP* create_HTTP();
16  HTTP* lecture_HTTP(char* http);
17  void free_HTTP(HTTP *http);
```

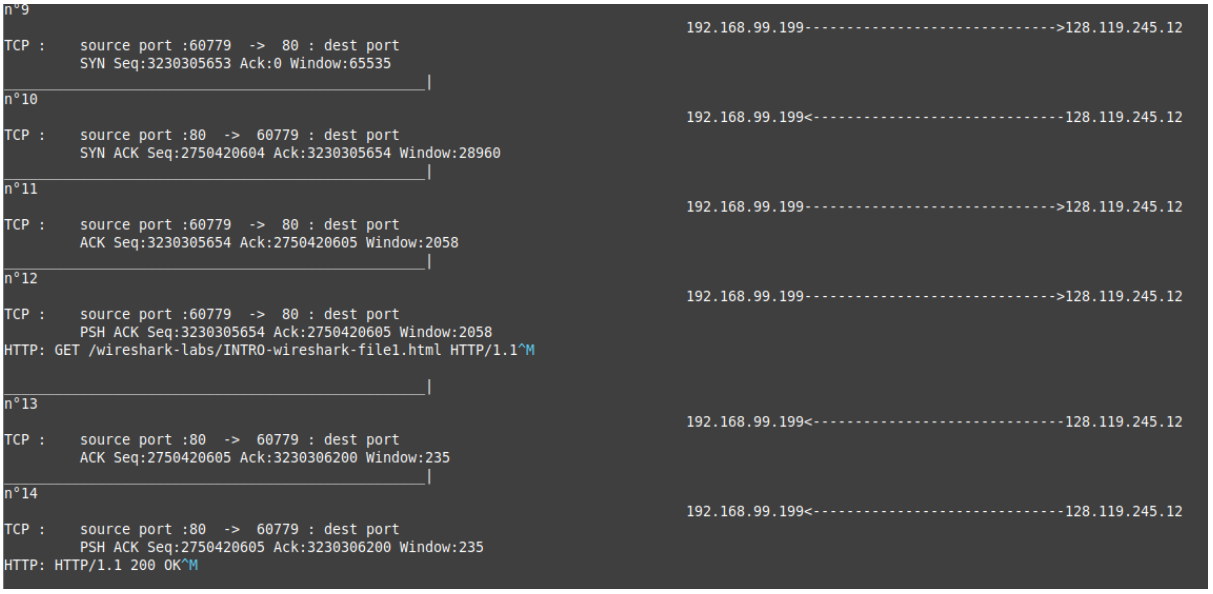
Pour HTTP, nous avons choisi de stocker uniquement la ligne de requête/réponse dans le champ ligne de notre structure, et les lignes d'entête et le corps de la requête dans le champ data. Nous avons pu faire ça grâce à la fonction lecture_HTTP, ou on a lu chaque caractère ASCII au format hexadécimal et on le concatène jusqu'à attendre la fin de la ligne de requête/réponse.

I. lecture.c et lecture.h

```
int lecture_paquet(FILE*f,char*c);
char* lecture_paquet_sans_frag(char *filename);
liste* lecture_fichier(char *filename,char** tabadd);
int decapsuler(char* paquet,filter* filtre,int n,FILE* f,char** tabadd,int n2);
void affichage_liste(liste *l,filter *f,char* filename,char **tabadd,int n2);
int addtableau(char* paquet, char** tabadd, int i);
void printFLOWCHART(char* source, char* dest, char** tabadd,FILE* f,int n2);
```

Dans ces fichiers, on a 7 fonctions qui sont indispensables pour la lecture et l'affichage de paquets. Dans un premier temps, on a la fonction lecture_paquet qui prend en paramètre un fichier et une chaîne de caractères. Cette fonction va stocker dans la chaîne c, ligne par ligne, tous les paquets. Cette fonction a été implémentée pour l'utiliser dans la fonction lecture_fichier, qui prend en paramètre le nom du fichier et renvoie la liste chaînée de paquets lus parmi le fichier passé en paramètre et qui modifie le tableau de chaîne de caractère afin de faire l'affichage en flow chart. Cette fonction utilise la fonction lecture_paquet et elle stocke chaque paquet dans le champ paquet de la liste chaînée de paquets (fichiers liste.c et liste.h). La fonction lecture_paquet_sans_frag, qui prend en paramètre le nom d'un fichier qui contient un seul paquet sans fragment offset, renvoie le paquet contenu dans le fichier stocké, sans espaces, dans une chaîne de caractères. La fonction decapsuler prends en paramètre une chaine de caractère représentant le paquet entier, un filtre lu préalablement (que nous allons expliquer plus en détail dans la section menu.c/menu.h), un nom de fichier qui sera le nom du fichier où l'on écrit notre flow graph, le tableau d'adresses lues pour le flow graph et n2 la taille de notre tableau d'adresse qui sert également pour le flow graph. cette fonction va successivement séparer les entêtes puis en analyser les valeurs pour finalement les filtrer à l'aide de la valeur du filtre ou bien les afficher, dans un premier temps dans le terminal sous forme simplifiée et avec la fonction printFLOWGRAPH pour la version visualiseur de

flots dans le fichier d'output. affichage_liste va donc être notre fonction pour décapsuler successivement chacun des paquets de notre liste chaînée de chaîne de caractères afin de faire une analyse de tout notre fichier source. La fonction addtableau nous permet de rajouter dans un tableau toutes les adresses IP ou bien MAC dans le cas de ARP, et ce avec gestion des doublons, cela nous permet d'avoir un affichage flow graph car nous associons un numéro de colonne à une adresse. La fonction printFLOWCHART nous permet de simuler la fonction flow graph de Wireshark dans un fichier .txt. En connaissant la taille de notre tableau, on le parcours jusqu'à trouver les indices de nos adresses source et adresses destinations, si l'indice i de la source est plus petit que l'indice j de la destination, alors on imprime i tabulations, puis l'adresse source, par la suite on imprime une flèche de longueur j-i allant vers la droite, arrivés à l'indice j on imprime la destination. Dans le cas inverse où i est supérieur à j, on imprime d'abord j tabulations, l'adresse destination, mais cette fois ci on imprime une flèche vers la gauche jusqu'à i et enfin l'adresse source. De cette manière nous avons bien un alignement des adresses dans le cas d'une connexion TCP.



```c
typedef struct _filter{
    int type;
    char* value;
} filter;

char* nomfichierlecture();
char* nomfichierecriture();
filter* create_filter();
filter* options();
void WireTacosBlink();
void free_filter(filter* f);
```

J. menu.c et menu.h

Le fichier menu possède des fonctions d'affichage et de lecture qui ne sont pas directement liées avec de l'analyse de trame, elle possède une structure de type filter qui nous permet de stocker en mémoire le type du filtre ainsi que la valeur de ce filtre stocké sous forme de chaîne de caractères. Notre fonction nomfichierlecture nous permet de lire dans le terminal le nom du fichier à lire pour l'analyse, elle vérifie également que le fichier soit bien présent dans le fichier et recommence jusqu'à ce qu'un nom de fichier valide soit entré. La fonction nomfichierecriture lit dans le terminal le nom du fichier d'output et rajoute systématiquement l'extension .txt afin d'assurer le bon format du fichier. create_filter nous permet d'allouer en mémoire une variable de type filter. La fonction options alloue un filtre et va alors afficher un menu dans lequel l'utilisateur peut naviguer afin de choisir son filtre, les filtres disponibles sont

- addresses MAC
- valeur du protocole encapsulé par ethernet
- addresses IP
- valeur du protocole encapsulé par IP
- valeur du type ARP
- valeur du type ICMP
- valeur du port TCP
- numéro de la trame dans le fichier

WireTacosBlink nous affiche notre logo clignotant et free_filter libère l'espace mémoire de notre filtre alloué

III. Conclusion

WireTacos dans la version telle que vous l'observez passe une grande partie de son temps à analyser les différents entêtes d'une trame, il suffirait de changer les différents affichages que nous faisons afin d'obtenir une fonction différente de Wireshark. Comme énoncé en introduction nous avons bien un code malléable. Une critique que nous pourrions faire serait sur le choix des structures de données choisies, notamment pour le tableau d'adresses que l'on accède souvent et qu'un arbre binaire complet que nous n'avons pas pu incorporer faute de temps. Une autre critique possible serait de modifier les filtres afin d'utiliser des requêtes SQL pour avoir plus de fonctionnalités de filtrage. Malgré tout, Wire Tacos reste parfaitement fonctionnel mais perfectionnable.