CEI – NETSEC #1

TECH documentation

La globalité du projet Wireshark a été développée en C. Nous n'utilisons pas de librairies pcap pour capturer, créer, importer ou exporter les packets. Tout a été réalisé manuellement.

I. Capture des packets depuis les Raw Sockets

Afin de capturer les packets défilant sur le réseau internet auquel est connecté l'ordinateur utilisant notre logiciel, nous utilisons l'objet socket, disponible dans le header <sys/socket.h>, connecté sur le réseau et prêt à capturer ses packets :

```
socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))
```

Nous passons ensuite cette socket à la fonction recvfrom qui prend en paramètre, en plus de la socket, un unsigned char *buffer qui chargera le contenu du packet en hexadécimal.

Une fois le buffer chargé, nous le passons en paramètre à nos fonctions qui permettent une lecture claire, mais aussi une création d'une liste des packets reçus.

Nous pourrons alors connaître toutes les informations du packet tel que son protocol Ethernet, son protocol IP, l'adresse IP d'où il provient et où il a été transmis, la taille qu'il fait, et bien d'autres…

II. Composition et création d'un packet

Une fois le packet capturé, nous allons pouvoir connaître ses informations, forger le packet, mais également le stocker dans une liste chaînée :

```c
typedef struct raw_packet_s {
  guint num;
  double time;
  int length;
  protocol_t proto;
  ethernet_header_t *eth;
  ip_header_t *ip;
  info_packet_t *info;
  data_dump_t *dump;
  struct raw_packet_s *next;
  struct raw_packet_s *prev;
} raw_packet_t;
```

Nous pouvons ici observer l'architecture principale de notre packet forging. C'est une structure comprenant toutes les informations du packet, reliée aux autres packets via les pointeurs next et prev.

Elle contient :

Un numéro (num) qui est l'index du packet.

Un timer (time) qui va savoir à quel moment a été enregistré le packet.

Une taille (length) qui mesure la taille du packet en octet.

Un protocole (proto) qui est une énumération désignant chaque type de packet que nous pouvons capturer :

```c
typedef enum protocol_e {
  TCP = 1,
  UDP = 2,
  ICMP = 3,
  ARP = 4,
  HTTP = 5,
  DNS = 6,
  Unknown = 0
} protocol_t;
```

Nous capturons les packets de protocole Ethernet IPv4 et ARP uniquement.

Les protocoles TCP, UDP, ICMP, HTTP et DNS sont de types IPv4.

Enfin, elle se constitue de 3 autres structures essentielles représentant l'architecture d'un packet de manière général :

Une structure Ethernet :

```c
typedef struct ethernet_header_s {
  char *src_addr;
  char *dest_addr;
  unsigned short proto;
} ethernet_header_t;
```

Nous pouvons y retrouver :

L'adresse source ethernet.

L'adresse de destination.

Mais aussi son protocol Ethernet, définit quelques lignes plus haut (IPv4 ou ARP).

Pour revenir à notre structure raw_packet_t principale, si le protocol Ethernet est IPv4, nous pouvons créer une struct IP qui détiendra les informations ci-dessous :

```
typedef struct ip_header_s {
  unsigned int version;
  unsigned int header_len; //en octet
  unsigned int service_type;
  unsigned short total_len;
  unsigned short id;
  unsigned int ttl;
  unsigned int proto;
  unsigned short checksum;
  char *src_ip;
  char *dest_ip;
} ip_header_t;
```

Les informations les plus importantes ici sont :

Le protocole IP (proto) qui peut être TCP, UDP, HTTP ou DNS.

L'adresse source IP et l'adresse de destination IP.

Enfin, pour tous les packets, nous pouvons avoir les informations propres à chaque protocoles grâce à cette structure :

```
typedef struct info_packet_s {
  tcp_header_t *tcp;
  udp_header_t *udp;
  icmp_header_t *icmp;
  arp_header_t *arp;
} info_packet_t;
```

Cette structure contient 4 autres structures en fonction du protocole IP venant d'être capturé.

Il est à savoir qu'un protocole HTTP est une surcouche au protocole TCP, il survient fréquemment lorsque le port source ou le port destination du packet TCP est égale à 80.

C'est la même chose pour le protocole DNS vis-à-vis du protocole UDP. Il est présent lorsque le port source ou le port destination est égale à 53.

Voici toutes les informations supplémentaires disponible dans un packet de protocole IP TCP :

```
typedef struct tcp_header_s {
  unsigned short src_port;
  unsigned short dest_port;
  unsigned long seq;
  unsigned long ack_seq;
  unsigned int len;
  unsigned int urg;
  unsigned int ack;
  unsigned int push;
  unsigned int reset;
  unsigned int sync;
  unsigned int fin;
  unsigned short window;
  unsigned short checksum;
  int urg_ptr;
} tcp_header_t;
```

Protocole IP UDP :

```
typedef struct udp_header_s {
  unsigned short src_port;
  unsigned short dest_port;
  unsigned short len;
  unsigned short checksum;
} udp_header_t;
```

Protocole IP ICMP :

```
typedef struct icmp_header_s {
  unsigned int type;
  unsigned int code;
  unsigned short checksum;
} icmp_header_t;
```

Protocole ARP :

```
typedef struct arp_header_s {
  unsigned short hrdw_f;
  unsigned short proto_f;
  unsigned char hrdw_len;
  unsigned char proto_len;
  unsigned short op;
} arp_header_t;
```

Enfin, nous stockons tout le code hexadécimal d'un packet, ainsi que sa valeur ascii dans une structure data_dump_t :

```
typedef struct data_dump_s {
  char *hexa;
  char *ascii;
} data_dump_t;
```

Le char *ascii est simplement une copie du buffer hexadécimal qui affiche chaque caractère pouvant être affiché et transforme les autres en point « . »

Connaissant toutes les informations d'un packet, nous pouvons aisément afficher leur contenu comme le ferait le logiciel original : Wireshark.
En plus de cela, nous avons également la capacité de filtrer les packets dès leur capture, ou à l'importation en fonction de leur protocole, de leur adresse IP ou bien de leur port IP.

III. Importation et Exportation de fichiers de capture pcap

Nous avons implémenté l'importation et l'exportation d'une capture de packet.

Elles sont pcap compatibles, et donc compatibles avec les fichiers wireshark.

Un fichier pcap est composé entièrement en hexadécimal, plus précisément en unsigned char, valeur allant de 0 à 255.

Il est composé d'un header global qui :

```
typedef struct pcap_hdr_s {
guint32 magic_number; /* magic number */
guint16 version_major; /* major version number */
guint16 version_minor; /* minor version number */
gint32 thiszone; /* GMT to local correction */
guint32 sigfigs; /* accuracy of timestamps */
guint32 snaplen; /* max length of captured packets, in octets */
guint32 network; /* data link type */
} pcap_hdr_t;
```

Et du code hexadécimal de chaque packet contenu dans le fichier. Chaque packet à un header personnel :

```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;         /* timestamp seconds */
        guint32 ts_usec;        /* timestamp microseconds */
        guint32 incl_len;       /* number of octets of packet saved in file */
        guint32 orig_len;       /* actual length of packet */
} pcaprec_hdr_t;
```