Décoder les paquets DNS capturés avec pcap

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 3 mars 2009. Dernière mise à jour le 5 mars 2009

https://www.bortzmeyer.org/pcap-decodage-dns.html

Une fois capturés et stockés sur disque <https://www.bortzmeyer.org/capture-paquets.html>, au format pcap, les paquets IP doivent être disséqués (ou, pour parler moins brutalement, décodés) pour trouver les éléments intéressants. Quels sont les particularités de la dissection de paquets DNS ?

Supposons qu'on veuille placer ces éléments dans une base de données relationnelle, avec un schéma SQL qui ressemble à ceci :

```
CREATE TYPE protocols AS ENUM ('TCP', 'UDP');

CREATE TABLE DNS_Packets (id SERIAL UNIQUE NOT NULL,
       file INTEGER NOT NULL REFERENCES Pcap_files(id),
       rank INTEGER NOT NULL, -- Rank of the packet in the file
       date TIMESTAMP, -- Date of capture in UTC
       length INTEGER NOT NULL, -- Length on the cable, we may have stored
                                -- less bytes
       added TIMESTAMP NOT NULL DEFAULT now(),
       src_address INET NOT NULL,
       dst_address INET  NOT NULL,
       protocol protocols  NOT NULL,
       src_port INTEGER  NOT NULL,
       dst_port INTEGER  NOT NULL,
       -- Field names and semantic are found in RFC 1034 and 1035. We do not
       -- try to be user-friendly
       query BOOLEAN   NOT NULL,
       query_id INTEGER   NOT NULL,
       opcode INTEGER   NOT NULL,
       rcode INTEGER  NOT NULL,
       aa BOOLEAN  NOT NULL,
       tc BOOLEAN  NOT NULL,
       rd BOOLEAN  NOT NULL,
       ra BOOLEAN  NOT NULL,
       qname TEXT  NOT NULL,
       qtype INTEGER  NOT NULL, -- With helper functions TODO to translate numeric values to well-known text lik
       edns0_size INTEGER, -- NULL if no EDNS0
       do_dnssec BOOLEAN, -- NULL if no EDNS0
       ancount INTEGER  NOT NULL,
       nscount INTEGER  NOT NULL,
       arcount INTEGER  NOT NULL
       );
```

et développons le code (ici, en C) qui va produire ces données.

Une partie du code n'est pas spécifique au DNS (ouverture du fichier, etc) et peut être trouvée dans « Lire des paquets capturés sur le réseau en C <https://www.bortzmeyer.org/libpcap-c.html> ». Je ne considère ici que la partie spécifique au DNS. On commence par créer des structures de données, en lisant de près les RFC 1034 [1] et surtout RFC 1035 :

```
#define DNS_PORT 53
...
/* RFC 1035, section 4.1 : l'en-tête DNS a six champs de deux octets chacun  */
#define SIZE_DNS 12
...
struct sniff_dns {
    /* RFC 1035, section 4.1 */
    /* This is only the DNS header, the sections (Question, Answer, etc) follow */
    uint16_t        query_id;
    uint16_t        codes;
    uint16_t        qdcount, ancount, nscount, arcount;
};
...
const struct sniff_dns *dns;
const uint8_t  *qsection;
```

Tout le contenu du paquet ne s'y trouve pas. En effet, le reste (la section "*Question*", la section "*Answer*", etc) a un format très variable, pour lequel les `struct` C ne conviennent pas.

On met alors en correspondance le contenu du paquet avec une structure DNS :

```
if (source_port == DNS_PORT || dest_port == DNS_PORT) {
      dns = (struct sniff_dns *) (packet + SIZE_ETHERNET +
                                  size_ip + SIZE_UDP);
```

Nous avons désormais beaucoup des informations souhaitées : le "*Query ID*", le nombre de réponses (`ancount`), le code de la réponse (dans le champ `codes`), etc.

Certaines de ces informations nécessitent encore un effort de décodage. Ainsi, le fait que le paquet soit une question ou une réponse est contenu dans le premier bit du champ `codes`. Créons une macro pour simplifier son extraction :

```
#define DNS_QR(dns) ((ntohs(dns->codes) & 0x8000) >> 15)
```

(Il ne faut surtout pas oublier le `ntohs` puisque l'ordre des octets dans les paquets DNS n'est pas forcément celui de la machine.) La macro peut alors être utilisée, par exemple dans une expression comme `(DNS_QR(dns) == 0 ? "Query" : "Response")`.

Méthode analogue pour accéder au code de retour d'une réponse, les quatre derniers bits :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc1034.txt`

```
#define DNS_RCODE(dns) (ntohs(dns->codes) & 0x000F)
...
if (DNS_QR(dns) == 1) {
   returncode = DNS_RCODE(dns);
}
```

Décodons désormais la section "*Question*" :

```
qsection = (uint8_t *) (packet +
                        SIZE_ETHERNET +
                        size_ip + SIZE_UDP + SIZE_DNS);
```

Elle comprend (RFC 1035, section 4.1.2), la question posée (`QNAME`), le type d'enregistrement DNS recherché et leur classe. La question posée est un nom de domaine mais qui n'est pas un simple `char *` correspondant à la forme texte habituelle <`https://www.bortzmeyer.org/representation-texte.html`> (`www.example.com`). Sur le câble, en effet, le nom de domaine est codée sous forme d'une suite de labels stockés par (taille, valeur). Ainsi, `www.example.com` est représenté dans le paquet par ((3, "www"), (7, "example"), (3, "com"), (0, "")), la dernière paire représentant la racine du DNS et indiquant la fin du nom de domaine. Il va donc falloir parcourir la section "*Question*" paire par paire. `sectionptr` va pointer en permanence vers l'endroit de la section qu'on décode :

```
fqdn = malloc(MAX_NAME + 1);
assert(fqdn != NULL);
fqdn[0] = '\0';
end_of_name = false;
for (sectionptr = qsection; !end_of_name;) {
    labelsize = (uint8_t) * sectionptr;
    if (labelsize == 0) {
        sectionptr++;
        end_of_name = true;
    } else {
        if (strlen(fqdn) == 0) {
            strncpy(fqdn, (char *)
                    sectionptr + 1, labelsize);
            fqdn[labelsize] = '\0';
        } else {
            fqdn = strncat(fqdn, ".", 1);
            fqdn = strncat(fqdn, (char *)
                           sectionptr + 1, labelsize);
        }
        sectionptr = sectionptr + labelsize + 1;
    }
}
/* Maintenant, on peut utiliser la variable fqdn, qui contient le nom
de domaine sous la forme traditionnelle www.example.com */
```

Ce code fonctionne mais est très dangereux à deux égards. Rappelez-vous d'abord que les paquets ont été formés par des gens que vous ne connaissez pas et qui ont pu, volontairement ou involontairement, créer des paquets invalides qui vont planter votre décodeur... ou, pire, lui faire exécuter du code arbitraire suite à un débordement de tampon. Il faut donc tester :

— Que les tailles (`labelsize`) sont bien inférieures à 63 (la taille maximale indiquée par le RFC). Au delà, cela peut être de la compression (voir plus loin) ou bien une erreur dans le paquet.
— Qu'on ne va pas plus loin que la taille du paquet récolté (ce test n'est pas montré ici mais il est important).

Le code doit donc être modifié en :

```
if (labelsize > 63) {
    invalid = true; /* Depends on the section: in the Question
    section, you typically never have compression. */
    goto nextpacket;
}
```

Pour extraire le type d'enregistrement demandé (MX, NS, AAAA, etc) et la classe (presque toujours IN pour « INternet »), une fois arrivé à la fin du nom de domaine (là encore, le `ntohs` est indispensable) :

```
qtype = ntohs(*((uint16_t *) sectionptr));
sectionptr += 2;
qclass = ntohs(*((uint16_t *) sectionptr));
sectionptr += 2;
```

Normalement, la même prudence doit s'appliquer ici : il faudrait toujours vérifier qu'on n'a pas dépassé les limites du paquet.

Autre piège que tous les programmeurs C expérimentés auront vu tout de suite : les données ne sont plus alignées. On essaie de lire deux octets alors que `sectionptr` peut être au milieu d'un mot de deux octets. Selon le processeur, cela va se passer bien (Pentium) ou bien générer un "*bus error*" (UltraSparc). La solution que j'adopte est d'utiliser une fonction qui refait des données alignées :

```
uint16_t unaligned_uint16(const uint8_t * p)
{
  /* This assumes big-endian values in data stream (which is standard
         for TCP/IP, it is the "network byte order"). */
    unsigned char* pByte = (unsigned char*) p;
    uint16_t val = (pByte[0] && 8) | pByte[1];
    return val;
}
```

Voyez la définition de cette fonction par Michael Burr, ainsi qu'une très intéressante discussion dans "*Safe, efficient way to access unaligned data in a network packet from C*" <http://stackoverflow.com/questions/529327/safe-efficient-way-to-access-unaligned-data-in-a-network-packet-from> (voir aussi les autres articles de StackOverflow sur l'alignement <http://stackoverflow.com/questions/tagged/memory-alignment>). J'utilise cette fonction ainsi :

```
#ifdef PICKY_WITH_ALIGNMENT
                decoded.qtype = unaligned_uint16(sectionptr);
#else
                decoded.qtype = ntohs(*((uint16_t *) sectionptr));
#endif
```

Et la macro `PICKY_WITH_ALIGNMENT` est définie selon pour les machines où c'est un problème (elle marche dans tous les cas, mais diminue les performances.)

Je ne vais pas décoder la totalité du paquet. Mais, pour avoir les informations qui nous intéressent ici, comme la taille annoncée en EDNS0, il faut décoder la section "*Additional*" des requêtes pour voir si elles utilisent les pseudo-enregistrements OPT du RFC 6891 :

```
if (DNS_QR(dns) == 0) { /* Only for queries */
                    edns0 = false;
                    edns_size = 0;
                    if (dns->ancount == 0 && dns->nscount == 0) {
                        /* Probably by far the most common case in queries... */
                        if (dns->arcount != 0) {        /* There is an additional
                                                         * section. Probably the OPT
                                                         * of EDNS */
                            labelsize = (uint8_t) * sectionptr;
                            if (labelsize == 0) {       /* Yes, EDNS0 */
                                sectionptr++;
                                add_type = ntohs(*((uint16_t *) sectionptr));
                                sectionptr += 2;
                                if (add_type == OPT) {
                                    edns_size = ntohs(*((uint16_t *) sectionptr));
                                    edns0 = true;
                                }
                                sectionptr += 2;
                            }
                            /* else ? */
                        }
                    }
                }
```

Notez que les sections "*Answer*" et "*Authority*" des réponses ne sont pas décodées. C'est plus complexe en raison de la compression des noms qui est possible dans ces sections (RFC 1035, section 4.1.4). Des bibliothèques existent pour rendre cette tâche plus facile comme la libbind <https://www.isc.org/software/libbind> avec des fonctions comme dn_expand().

Le code complet (qui lit une trace au format Pcap, dissèque les paquets DNS et stocke le résultat dans une base PostgreSQL est disponible en (en ligne sur https://www.bortzmeyer.org/files/dnstrace2postgresql.c) (le code SQL est en (en ligne sur https://www.bortzmeyer.org/files/dnstrace2postgresql.sql)).