Parsifal : une solution pour écrire rapidement des *parsers* binaires robustes et efficaces

Olivier Levillain[1,2]

1: Agence nationale de la sécurité des systèmes d'information (ANSSI)
2: Télécom Sud Paris

Dans le cadre de ses activités d'expertise, le laboratoire sécurité des réseaux et protocoles de l'ANSSI est amené à étudier divers protocoles de communication. L'étude fine de ces protocoles passe par l'utilisation de *parsers* (ou dissecteurs) permettant d'analyser les messages binaires échangés lors d'une exécution du protocole. L'expérience a montré qu'il fallait disposer d'outils robustes et maîtrisés pour étudier et comprendre les comportements d'un protocole donné, en particulier pour en détecter et caractériser les anomalies. En effet, les implémentations disponibles sont parfois limitées (refus de certaines options), laxistes (acceptation silencieuse de paramètres erronés) ou fragiles (terminaison brutale du programme pour des valeurs inattendues, qu'elles soient licites ou non). Ce constat nous a conduit au développement d'outils, l'objectif étant de développer *rapidement* des dissecteurs *robustes* et *performants*. Ce document décrit brièvement Parsifal, une implémentation générique de *parsers* binaires reposant sur le pré-processeur `camlp4` d'OCaml.

Afin de mieux comprendre un protocole et la manière dont il est utilisé *in vivo*, le laboratoire s'intéresse notamment à l'analyse de grands volumes de données issus de mesures réalisées sur internet. Le point de départ de nos travaux sur les *parsers* binaires est un ensemble important de traces réseau contenant des échanges suivant le protocole TLS (*Transport Layer Security* [3]) mises à disposition par l'EFF (*Electronic Frontier Foundation*) [1]. Ces mesures ont fait l'objet d'une publication [2]. L'analyse de ces données pose plusieurs difficultés. Tout d'abord, les fichiers à analyser représentent un volume conséquent (180 Go dans le cas de notre analyse de TLS). Ensuite, les informations à extraire sont contenues dans des messages de structures complexes. Enfin, il s'agit de données brutes, non filtrées, dont la qualité, voire l'innocuité, laisse parfois à désirer.

Description de Parsifal

Parsifal est issu des besoins identifiés et de l'expérience acquise dans l'écriture de *parsers* pour des formats binaires. Il s'agit d'une implémentation générique de *parsers* reposant sur un pré-processeur `camlp4` et sur une bibliothèque auxiliaire.

Le concept de base de Parsifal est la définition de « types enrichis », les $\mathcal{PT}$ypes, qui sont simplement des types OCaml quelconques pour lesquels certaines fonctions sont fournies. Ainsi, un $\mathcal{PT}$ype est défini par un type OCaml `t` décrivant le contenu à *parser*, par une fonction pour disséquer les objets depuis une chaîne de caractères (`parse_t`) et par des fonctions pour exporter les objets sous forme binaire (`dump_t`) ou dans une représentation haut niveau utile aux fonctions d'affichage (`value_of_t`).

On peut distinguer trois sortes de $\mathcal{PT}$ypes. Tout d'abord, la bibliothèque standard fournit des $\mathcal{PT}$ypes de base (entiers, chaînes de caractère, listes, etc.). Ensuite, il estpossible de construire des $\mathcal{PT}$ypes à partir de mots clés tels que `struct`, `union`, `enum` ; pour ceux-ci, une description suffit au pré-processeur pour générer automatiquement le type OCaml et les fonctions correspondantes. Enfin, dans certains cas, il est nécessaire d'écrire le type OCaml et les fonctions `parse_t`, `dump_t` et `value_of_t`

à la main, pour gérer des cas particuliers. Pour illustrer les deux premiers types de $\mathcal{PT}$ypes, voici une implémentation rudimentaire des messages TLS à l'aide de Parsifal :

```
enum tls_content_type (8, Exception) =
  | 0x14 -> CT_ChangeCipherSpec      | 0x15 -> CT_Alert
  | 0x16 -> CT_Handshake             | 0x17 -> CT_ApplicationData

union record_content (Unparsed_Record) =
  | CT_Alert             -> Alert of array(2) of uint8
  | CT_Handshake         -> Handshake of binstring
  | CT_ChangeCipherSpec -> ChangeCipherSpec of uint8
  | CT_ApplicationData  -> ApplicationData of binstring

struct tls_record = {
  content_type : tls_content_type;
  record_version : tls_version;
  record_content : container[uint16] of record_content (content_type)
}
```

Le dernier bloc de code décrit ce qu'est un *record* TLS, un enregistrement (décrit à l'aide du mot clé `struct`) contenant quatre champs : le type du contenu, la version du protocole, la taille du contenu et le contenu lui-même. Pour le premier champ, il existe 4 types de contenu, qui sont décrits par l'énumération `tls_content_type` (anoncée par le mot clé `enum` du premier bloc). Cette valeur est stockée sur un entier 8 bits, et si la lecture de ce champ donne une valeur non énumérée, une exception sera levée ; c'est le sens des paramètres de l'énumération (`8` et `Exception`).

La version TLS est stockée sur 16 bits : on utilise donc le $\mathcal{PT}$ype prédéfini `uint16`. Comme il existe certaines versions connues, on pourrait utiliser une énumération ici également, avec un comportement plus laxiste face aux valeurs inconnues (ajout d'un constructeur avec `UnknownVal`) :

```
enum tls_version (16, UnknownVal UnknownVersion) =
  | 0x0002 -> SSLv2       | 0x0300 -> SSLv3
  | 0x0301 -> TLSv1       | 0x0302 -> TLSv1_1
  | 0x0303 -> TLSv1_2
```

Les deux derniers champs du *tls_record* sont décrits ensemble par un conteneur dont la longueur, variable, tient sur 16 bits. Le contenu du message lui-même est décrit par le $\mathcal{PT}$ype `record_content`, qui prend un argument (`content_type`). En effet, `record_content` est une `union`, dont le contenu dépend d'un discriminant, ici le type de contenu. Par exemple, une alerte contient 2 octets.

Bilan de deux ans d'écriture de *parsers* binaires

Après avoir écrit plusieurs implémentations dans différents langages (Python, C++, OCaml), nous avons développé Parsifal, une implémentation générique de *parsers* binaires reposant sur un pré-processeur, qui remplit nos besoins : possibilité d'exprimer des formats complexes, rapidité d'écriture, robustesse et performances. Plusieurs protocoles réseau ont déjà été (au moins partiellement) décrits à l'aide de Parsifal (TLS, DNS, BGP...), ainsi que plusieurs formats de fichiers (TAR, PNG, JPG...).

Cette courte présentation de Parsifal n'a pas fait état d'autres constructions pratiques pour le développeur : gestion des structures ASN.1 DER, conteneurs personnalisés, champs de bits... Le projet est disponible en tant que logiciel libre sur `https://github.com/ANSSI-FR/parsifal`.

Références

[1] P. Eckersley and J. Burns. An Observatory for the SSLiverse, Talk at Defcon 18, 2010.

[2] O. Levillain, A. Ébalard, H. Debar, and B. Morin. One Year of SSL Measurement. ACSAC, 2012.

[3] T. Dierks and E. Rescorla. The Transport Layer Security Protocol Version 1.2. RFC 5246, 2008.