Parsifal : écriture rapide de *parsers* robustes et efficaces

Olivier Levillain

Résumé

Dans le cadre de ses activités d'expertise, le laboratoire sécurité des réseaux et protocoles (LRP) 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, de confiance permettant d'analyser les messages échangés lors d'une exécution du protocole.

L'expérience a montré qu'il était nécessaire de disposer d'implémentations indépendantes et robustes pour étudier et comprendre les comportements d'un protocole donné, en particulier pour détecter et caractériser les anomalies. En effet, pour un protocole donné, les implémentations disponibles de clients, serveurs ou dissecteurs 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 a conduit au développement d'outils, l'objectif étant de développer *rapidement* des dissecteurs *robustes* et *efficaces*. Pour cela, plusieurs langages de programmation ont été successivement utilisés (C, C++, python, OCaml). L'objet de ce document est de présenter l'une de ces implémentations, reposant sur le pré-processeur `camlp4` d'OCaml.

Historique du projet

Motivation initiale : analyser de nombreuses réponses TLS

Le point de départ de ces travaux était l'analyse d'une quantité importante (180 Go) de données récoltées concernant le protocole SSL/TLS. Afin de pouvoir interpréter correctement les différents types de réponses, il est nécessaire de comprendre les différentes versions du protocole, qui apportent chacune des subtilités dans les messages échangés. Parmi les messages pertinents pour ce type d'étude, le message `Certificate` contient la chaîne de certification permettant l'authentification du serveur. Elle est composée de certificats X.509 [1], ce qui apporte une complexité supplémentaire pour la dissection des messages.

Le laboratoire sécurité des réseaux et protocoles a mené des travaux sur de nombreuses traces HTTPS depuis deux ans. Les données utilisées proviennent de résultats mis à dispositions par l'Electronic Frontier Foundation [2, 3, 4] en 2010 et de collectes effectuées en 2011 dans le cadre du projet européen VIS-SENSE [5]. Les traces analysées contiennent une très grande diversité de réponses, parfois incohérentes ou non conformes. Face à cet imposant corpus, il était difficile d'utiliser des outils existants pour extraire de manière fiable l'information pertinente pour les analyses.

Des outils spécifiques, maîtrisés par le laboratoire, ont donc été développés pour analyser ce grand volume de données.

Démarche de développement des outils

Le premier *parser* a été écrit en Python. Nous avons ainsi obtenu rapidement un prototype pour extraire les premiers éléments des données. Cependant, ce premier programme s'est révélé trop lent face au volume à traiter.

Une seconde implémentation a donc vu le jour, en C++. Celle-ci reposait sur les *templates* et la programmation objet, et permettait d'obtenir un dissecteur flexible et efficace, mais au prix d'une grande quantité de code à écrire et d'erreurs de programmation parfois difficiles à diagnostiquer (fuites mémoire, erreurs de segmentation).

Pour pallier ce manque de robustesse, le développement d'une troisième version des outils a été entreprise, en OCaml. Afin de conserver la flexibilité imaginée pour le second prototype, un langage spécifique a été développé pour décrire les structures à disséquer. Les outils résultants étaient expressifs, efficaces et plus fiables que la version précédente. Malheureusement, à l'usage, si l'extensibilité était réelle, elle nécessitait des développements fastidieux.

Finalement, une nouvelle implémentation en OCaml a été réalisée pour réunir l'ensemble des qualités recherchées pour le développement de *parsers*. Cette quatrième mouture, baptisée Parsifal, résulte de la fusion entre le langage spécifique de description des objets analysés et le langage de programmation : on utilise directement le langage OCaml pour les deux usages. Pour cela, le préprocesseur `camlp4` est employé pour générer automatiquement des types et des fonctions à partir de descriptions brèves des structures à disséquer.

Pour résumer, les propriétés qui semblent souhaitables pour un *parser* sont les suivantes : l'expressivité du langage utilisé, la robustesse de l'outil obtenu et l'efficacité du code produit. Parmi les *parsers* présentés ci-dessus, seul Parsifal présente de bonnes caractéristiques dans tous les domaines.

Quelques exemples

Parsifal fournit les types suivants : les types scalaires de base (entiers, chaînes de caractères), les listes et conteneurs, quelques types utiles (`magic`, `ipv4`, etc.) et de quoi écrire des grammaires ASN.1. Pour cela, l'implémentation repose sur des pré-processeurs `camlp4`, qui comportent environ 1000 lignes de code et des bibliothèques annexes pour implémenter les types de base, l'ASN.1 et toute les rouages internes, sur 3000 lignes de code.

Plusieurs formats et protocoles suivants ont été implémentés à partir de ces briques de base : les certificats X.509 avec leurs nombreuses extensions (900 lignes), les messages du protocole TLS (1000 lignes), les messages du protocole MRT (300 lignes), le format TAR (moins de 100 lignes), une première version du format PE et un support rudimentaire de PCAP, IP, TCP, UDP et DNS.

Pour donner une idée de la manière d'écrire du code avec Parsifal, le listing ci-dessous reproduit des extraits d'une implémentation simpliste du format TAR :

```
enum file_type =
   | 0 -> NormalFile
   | 0x30 -> NormalFile
   | 0x31 -> HardLink
   | 0x32 -> SymbolicLink
   | 0x33 -> CharacterSpecial
   | 0x34 -> BlockSpecial
   | 0x35 -> Directory
   | 0x36 -> FIFO
   | 0x37 -> ContiguousFile
```

```
struct tar_header = {
  file_name : string(100);
  file_mode : string(8);
  [...]
  file_type : file_type;
  linked_file : string(100);
  ustar_magic : magic("ustar");
  [...]
}
```

Les mots-clés `enum` et `struct`[1] permettent d'écrire des descriptions à partir desquelles Parsifal génère automatiquement les types OCaml concrets `t` et un ensemble de fonctions : `parse_t` pour construire un objet à partir d'une chaîne de caractères, `dump_t` pour produire une version binaire à partir d'un objet, et `value_of_t` pour obtenir une représentation abstraite de l'objet, que l'on peut ensuite afficher de manière simple ou explorer.

Par exemple, l'outil `x509show`, qui utilise directement les types et fonctions générées par les descriptions X.509, permet les manipulations suivantes :

```
% x509show /etc/ssl/AC_RACINE_NEW.pem -g "tbsCertificate.subject.@CN"
"AC RACINE"
% x509show /etc/ssl/AC_RACINE_NEW.pem -g "tbsCertificate.subject.@C"
"FR"
% x509show /etc/ssl/AC_RACINE_NEW.pem -g "tbsCertificate.extensions.*.extnID"
["basicConstraints (2.5.29.19)", "authorityKeyIdentifier (2.5.29.35)",
 "subjectKeyIdentifier (2.5.29.14)", "keyUsage (2.5.29.15)"]
```

Conclusion et perspectives

Parsifal est en projet en cours de développement, qui a déjà fait ses preuves pour analyser certains protocoles réseau ou formats de fichier. Parsifal permet de générer des dissecteurs réunissant toutes les propriétés recherchées (rapidité de développement, robustesse et efficacité des programmes).

Deux projets écrits en Python présentent des similarités avec les travaux exposés ici : `scapy` [6], une boîte à outils pour manipuler les paquets réseau et `hachoir` [7], une bibliothèque pour écrire rapidement des dissecteurs pour des formats de fichiers.

Parsifal est aujourd'hui suffisamment mûr pour être partagé avec une communauté. Même si le coeur du moteur est suffisant pour de nombreux usages, les développements du socle de Parsifal continuent pour gérer toujours plus de cas : gestion simple des champs de bits, gestion des types binaires récursifs (tels que PKCS#7) ou encore intégration d'un véritable langage de description et d'analyse interactive.

L'autre axe de développement est l'ajout ou l'amélioration du support de nouveaux formats de fichiers et protocoles : DNS, OCSP, etc. Enfin, Parsifal sert aussi de base à des outils d'analyse de certificats et de connexions TLS, qui font aussi l'objet d'un développement actif.

1. Parsifal offre d'autres constructions telles que les `union`, les `alias` et les constructeurs spécifiques à ASN.1.

Remerciements

Parsifal n'aurait pas vu le jour sans le soutien et les conseils de d'Eric, Pierre, Guillaume et Arnaud (E), ainsi que les boutades de Vincent, Benjamin, Loïc et Arnaud (F).

Références

[1] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[2] Electronic Frontier Foundation. The EFF SSL Observatory. `https://www.eff.org/observatory`, 2010-2012.

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

[4] P. Eckersley and J. Burns. Is the SSLiverse a safe place ?, Talk at 27C3, 2010.

[5] Olivier Levillain, Arnaud Ébalard, Hervé Debar, and Benjamin Morin. One Year of SSL Measurement. In *ACSAC*, 2012.

[6] P. Biondi and the Scapy community. Scapy. `http://www.secdev.org/projects/scapy/`, 2003-2012.

[7] V. Stinner. Hachoir. `https://bitbucket.org/haypo/hachoir/wiki/Home`, 2009-2012.

Remarque à l'attention du comité de programme

Si cette proposition est retenue, la présentation rappellera brièvement l'historique du projet, puis les avantages et inconvénients des différents langages utilisés pour l'écriture de *parsers*. Enfin, je terminerai par une démonstration de l'outil.