Parsifal, ou comment écrire rapidement des *parsers* robustes et efficaces

Olivier Levillain

ANSSI

5 juin 2013

Contexte

- Pour comprendre un format ou un protocole, le mieux est de l'implémenter
- Comme souvent, le diable se cache dans les détails
  - encodage des entiers en ASN.1 ou en protobuf
  - *endianness* des champs, jusqu'à l'ordre de remplissage d'un octet bit-à-bit
  - sémantique des protocoles et spécifications floues
- Les *parsers* binaires sont une brique de base de toute implémentation
- Quelques vulnérabilités liées à des *parsers*
  - `libpng` : CVE-2011-3045 et CVE-2011-3026
  - `libtiff` : CVE-2012-5581, CVE-2012-4447 et CVE-2012-1173
  - `wireshark` : CVE-2012-4048, CVE-2012-4296...

Outils existants pour analyser des formats binaires

- Wireshark
- Scapy
- Hachoir

- Avantages
  - outils existants
  - de nombreux protocoles réseau et formats de fichiers déjà implémentés
  - Scapy (et Hachoir) facilement extensible

- Limitations
  - deux outils sont limités au réseau
  - Wireshark difficile à étendre
  - lenteur du python

Cas réel : analyse de données SSL

- Analyse de mesures SSL (article publié à ACSAC 2012)
  - pour chaque hôte contacté, réponse du serveur à un `ClientHello` TLS sur le port 443
  - 140 Go de données brutes
- Problème pour disséquer toutes ces données
  - données corrompues
  - protocole autre que SSL/TLS (en général HTTP ou SSH)
  - erreurs plus subtiles dans les messages

Interlude concernant SSL

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `DHE-RSA-AES128-SHA` ?

Interlude concernant SSL

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `DHE-RSA-AES128-SHA` ?

A `AES128-SHA`

Interlude concernant SSL

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `DHE-RSA-AES128-SHA` ?

A `AES128-SHA`

B `DHE-RSA-AES128-SHA`

Interlude concernant SSL

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `DHE-RSA-AES128-SHA` ?

A `AES128-SHA`

B `DHE-RSA-AES128-SHA`

C une alerte

Interlude concernant SSL

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `DHE-RSA-AES128-SHA` ?

A `AES128-SHA`

B `DHE-RSA-AES128-SHA`

C une alerte

D la réponse D (`RC4_MD5`)

Historique des outils

Pour traiter ce volume de données, plusieurs *parsers* TLS ont été développés

- Python : rapide à écrire, mais lent à l'exécution
- C++ (avec *templates* et des objets) : flexible, rapide, mais verbeux et pénible à mettre au point
- OCaml avec un préprocesseur : tous les indicateurs au vert

Parsifal : plaquette publicitaire

- Écriture de *parsers* grâce à du code **concis**
- **Efficacité** des programmes produits
- **Robustesse** des outils développés
- Méthodologie de développement adaptée à l'écriture **incrémentale** de *parsers* flexibles

Parsifal : plaquette publicitaire

- Écriture de *parsers* grâce à du code **concis**
- **Efficacité** des programmes produits
- **Robustesse** des outils développés
- Méthodologie de développement adaptée à l'écriture **incrémentale** de *parsers* flexibles

- Parsifal permet aussi de *dumper* les objets décrits
- Exemple : client DNS en 200 lignes

Parsifal : plaquette publicitaire

- Écriture de *parsers* grâce à du code **concis**
- **Efficacité** des programmes produits
- **Robustesse** des outils développés
- Méthodologie de développement adaptée à l'écriture **incrémentale** de *parsers* flexibles

- Parsifal permet aussi de *dumper* les objets décrits
- Exemple : client DNS en 200 lignes

- Objectifs de Parsifal
  - outils d'analyse maîtrisés
  - brique de base pour des outils de dépollution

Exemple : structure d'une image PNG (1/3)

```
struct png_file = {
  png_magic : magic("\x89\x50\x4e\x47\x0d\x0a\x1a\x0a");
  png_content : binstring;
}
```

Exemple : structure d'une image PNG (1/3)

```
struct png_file = {
  png_magic : magic("\x89\x50\x4e\x47\x0d\x0a\x1a\x0a");
  png_content : binstring;
}

let input = input_of_filename "sstic.png" in
let png = parse_png_file input in
print_value (value_of_png_file png)
```

Exemple : structure d'une image PNG (1/3)

```
struct png_file = {
  png_magic : magic("\x89\x50\x4e\x47\x0d\x0a\x1a\x0a");
  png_content : binstring;
}

let input = input_of_filename "sstic.png" in
let png = parse_png_file input in
print_value (value_of_png_file png)
```

Démo : notre premier *parser* PNG

Exemple : structure d'une image PNG (2/3)

```
struct png_file = {
  png_magic : magic("\x89\x50\x4e\x47\x0d\x0a\x1a\x0a");
  chunks : list of png_chunk;
}
```

Exemple : structure d'une image PNG (2/3)

```
struct png_file = {
  png_magic : magic("\x89\x50\x4e\x47\x0d\x0a\x1a\x0a");
  chunks : list of png_chunk;
}

struct png_chunk = {
  chunk_size : uint32;
  chunk_type : string(4);
  data : binstring(chunk_size);
  crc : uint32;
}
```

Exemple : structure d'une image PNG (2/3)

```
struct png_file = {
  png_magic : magic("\x89\x50\x4e\x47\x0d\x0a\x1a\x0a");
  chunks : list of png_chunk;
}

struct png_chunk = {
  chunk_size : uint32;
  chunk_type : string(4);
  data : binstring(chunk_size);
  crc : uint32;
}
```

Démo : résultat du préprocesseur

Exemple : structure d'une image PNG (2/3)

```
struct png_chunk = {
   chunk_size : uint32;
   chunk_type : string(4);
   data : container(chunk_size) of chunk_content(chunk_type);
   crc : uint32;
}
```

Exemple : structure d'une image PNG (2/3)

```
struct png_chunk = {
  chunk_size : uint32;
  chunk_type : string(4);
  data : container(chunk_size) of chunk_content(chunk_type);
  crc : uint32;
}

union chunk_content [enrich] (UnparsedChunkContent) =
| "IHDR" -> ImageHeader of image_header
| "IDAT" -> ImageData of binstring
| "IEND" -> ImageEnd
| "PLTE" -> ImagePalette of list of array(3) of uint8

struct image_header = {
  ...
}
```

Exemple : structure d'une image PNG (2/3)

```
struct png_chunk = {
  chunk_size : uint32;
  chunk_type : string(4);
  data : container(chunk_size) of chunk_content(chunk_type);
  crc : uint32;
}

union chunk_content [enrich] (UnparsedChunkContent) =
| "IHDR" -> ImageHeader of image_header
| "IDAT" -> ImageData of binstring
| "IEND" -> ImageEnd
| "PLTE" -> ImagePalette of list of array(3) of uint8

struct image_header = {
  ...
}
```

Démo : *parser* enrichi.

Parsifal : présent...

Fonctionnalités non présentées

- Conteneurs (base64, zlib, etc.)
- Manipulation simple des objets
  - `x509show -g "**.extnID" cert.pem`

Formats implémentés :

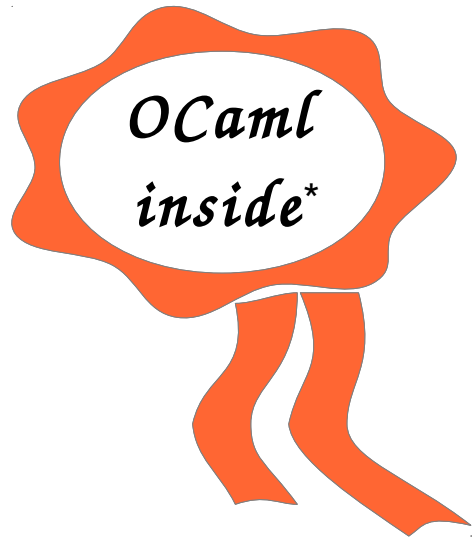
X.509	description assez complète
SSL/TLS	beaucoup de messages décrits
PE	code à intégrer
Kerberos	messages PKINIT à intégrer
TAR	tutoriel
DNS	tutoriel + `picodig`
PNG	tutoriel + stage en cours
PCAP	support rudimentaire PCAP/IP/TCP/UDP

Parsifal :... et futur

- Stabilisation v0.2 en cours
- Consolider la documentation
- Perspectives
  - assainissement PDF
  - animation des protocoles (TLS)

Questions ?

Merci de votre attention.



* mais ne protège pas des XSS

https://github.com/ANSSI-FR/parsifal