

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

**Olivier Levillain**

ANSSI

8 janvier 2014

# Contexte

- ▶ Pour comprendre un format ou un protocole, le mieux est de l'implémenter
- ▶ Chaque format/protocole repose sur des structures binaires qui lui sont propres
- ▶ 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...

## Cas réel : analyse de données SSL

SSL/TLS est un protocole réseau assurant la confidentialité, l'intégrité et l'authentification des parties

- ▶ Analyse de captures réseau avec des messages SSL
  - ▶ 180 Go de données brutes
  - ▶ corpus intéressant, car c'est ce que *voit* votre navigateur en permanence

## Cas réel : analyse de données SSL

SSL/TLS est un protocole réseau assurant la confidentialité, l'intégrité et l'authentification des parties

- ▶ Analyse de captures réseau avec des messages SSL
  - ▶ 180 Go de données brutes
  - ▶ corpus intéressant, car c'est ce que *voit* votre navigateur en permanence
- ▶ Problèmes pour disséquer toutes ces données
  - ▶ format de message complexe
  - ▶ 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 `camlp4` : tous les indicateurs au vert

## Parsifal : plaquette publicitaire

- ▶ Écriture de *parsers* grâce à du code **concis**
- ▶ **Efficacité** des programmes produits
- ▶ **Robustesse** des outils développés
- ▶ Développement **incrémental** de *parsers* flexibles

## Parsifal : plaquette publicitaire

- ▶ Écriture de *parsers* grâce à du code **concis**
- ▶ **Efficacité** des programmes produits
- ▶ **Robustesse** des outils développés
- ▶ Développement **incrémental** de *parsers* flexibles
  
- ▶ Parsifal permet de décrire des structures
- ▶ Génération des fonctions *parse* et *dump*
- ▶ 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
- ▶ Développement **incrémental** de *parsers* flexibles
  
- ▶ Parsifal permet de décrire des structures
- ▶ Génération des fonctions *parse* et *dump*
- ▶ Exemple : client DNS en 200 lignes
  
- ▶ Usages possibles de Parsifal
  - ▶ outils d'analyse robustes maîtrisés
  - ▶ brique de base pour des outils de dépollution
  - ▶ implémentations sécurisées de protocoles

## 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 "image.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 "image.png" in  
let png = parse_png_file input in  
print_value (value_of_png_file png)
```

Sortie du programme :

```
value {  
    png_magic: 89504e470d0a1a0a (8 bytes)  
    png_content: 0000000d49484... (264 bytes)  
}
```



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

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

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

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

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

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

Sortie du programme :

```
value {
  png_magic: 89504e470d0a1a0a (8 bytes)
  chunks {
    chunks[0] {
      chunk_size: 13 (0x0000000d)
      chunk_type: "IHDR" (4 bytes)
      data: 00000014000000160403000000 (13 bytes)
      crc: 846176565 (0x326fa135)
    }
    ... 4 autres chunks ...
  }
}
```

# Constructions offertes par Parsifal

- ▶ `struct` permet de décrire des enregistrements

# Constructions offertes par Parsifal

- ▶ `struct` permet de décrire des enregistrements
- ▶ `enum` pour les énumérations typées

# Constructions offertes par Parsifal

- ▶ `struct` permet de décrire des enregistrements
- ▶ `enum` pour les énumérations typées
- ▶ `union` pour des types dépendant d'un discriminant

# Constructions offertes par Parsifal

- ▶ `struct` permet de décrire des enregistrements
- ▶ `enum` pour les énumérations typées
- ▶ `union` pour des types dépendant d'un discriminant
- ▶ `asn1_*` pour décrire des structures ASN.1 (comme les certificats)

# Constructions offertes par Parsifal

- ▶ `struct` permet de décrire des enregistrements
- ▶ `enum` pour les énumérations typées
- ▶ `union` pour des types dépendant d'un discriminant
- ▶ `asn1_*` pour décrire des structures ASN.1 (comme les certificats)
- ▶ gestion des champs de bits



# Constructions offertes par Parsifal

- ▶ `struct` permet de décrire des enregistrements
- ▶ `enum` pour les énumérations typées
- ▶ `union` pour des types dépendant d'un discriminant
- ▶ `asn1_*` pour décrire des structures ASN.1 (comme les certificats)
- ▶ gestion des champs de bits
- ▶ notion de conteneurs pour gérer des transformations
  - ▶ `ztext` : `zlib_container of string`;

# Constructions offertes par Parsifal

- ▶ `struct` permet de décrire des enregistrements
- ▶ `enum` pour les énumérations typées
- ▶ `union` pour des types dépendant d'un discriminant
- ▶ `asn1_*` pour décrire des structures ASN.1 (comme les certificats)
- ▶ gestion des champs de bits
- ▶ notion de conteneurs pour gérer des transformations
  - ▶ `ztext` : `zlib_container of string`;
- ▶ et de nombreux types de bases prédéfinis...

## Exemples de formats décrits

Format	%	Remarques
X.509	90 %	Encodage DER automatisé
SSL/TLS	60 %	Outils d'analyse de traces + automate (en cours)
PCAP	25 %	Support rudimentaire du format de traces réseau
TAR	90 %	Tutoriel
DNS	75 %	<i>bit fields</i> , contexte pour la <i>compression</i>
PNG	80 %	Compression DEFLATE
JPG	30 %	Format moins serein que PNG
PE, Kerberos, OpenPGP, PKCS#1, PKCS#7...		
<i>Firmware UEFI, PDF...</i>		

# Conclusion

Bilan après deux ans de développement :

- ▶ Parsifal permet de décrire **rapidement** des formats **complexes**
- ▶ Les programmes obtenus sont **robustes** et **efficaces**
- ▶ Le code est facilement **réutilisable**
  
- ▶ Prise en main relativement facile
  
- ▶ Code disponible sur GitHub
  
- ▶ N'hésitez pas à venir en discuter !

# Questions ?

Merci de votre attention

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

`olivier.levillain@ssi.gouv.fr`