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

Olivier Levillain

Télécom SudParis

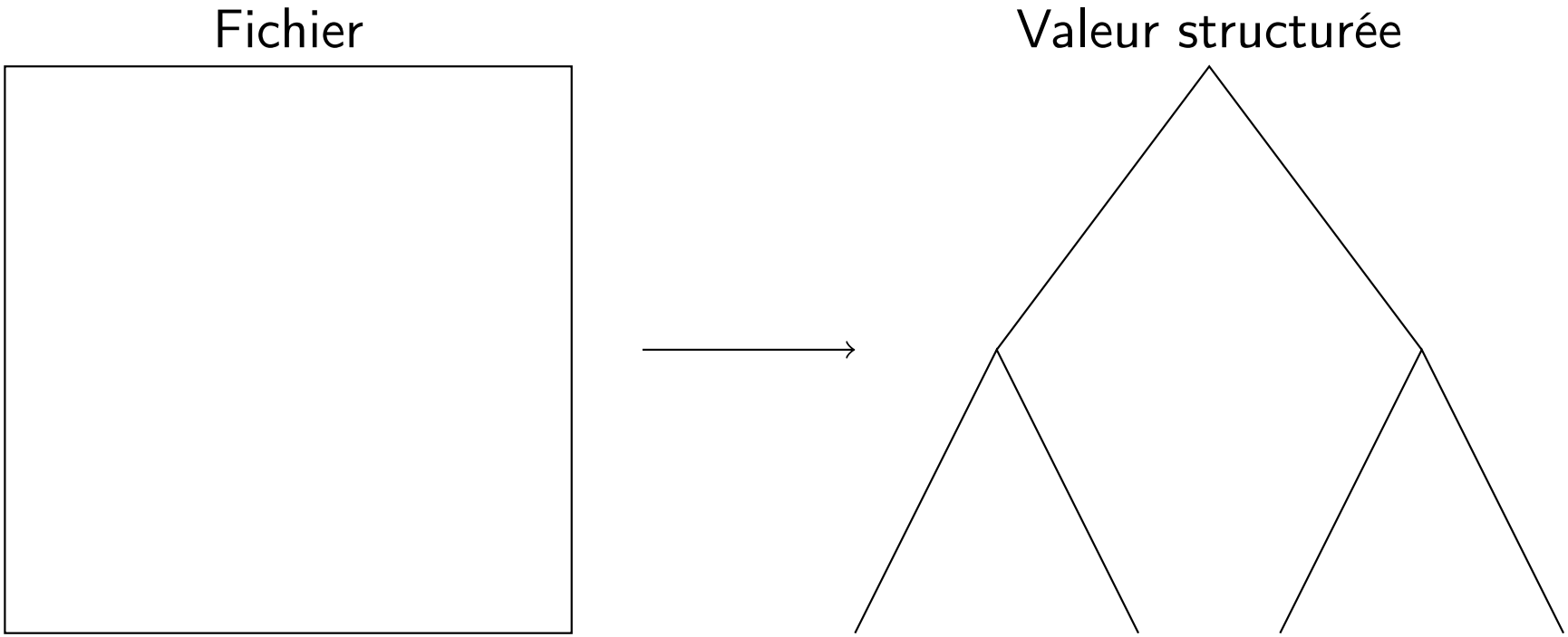
Journée LTP - Automne 2018

Motivation initiale

Parsifal

Limitations et nouvelles motivations : vers Parifal v2 ?

Qu'est-ce qu'un *parser* ?

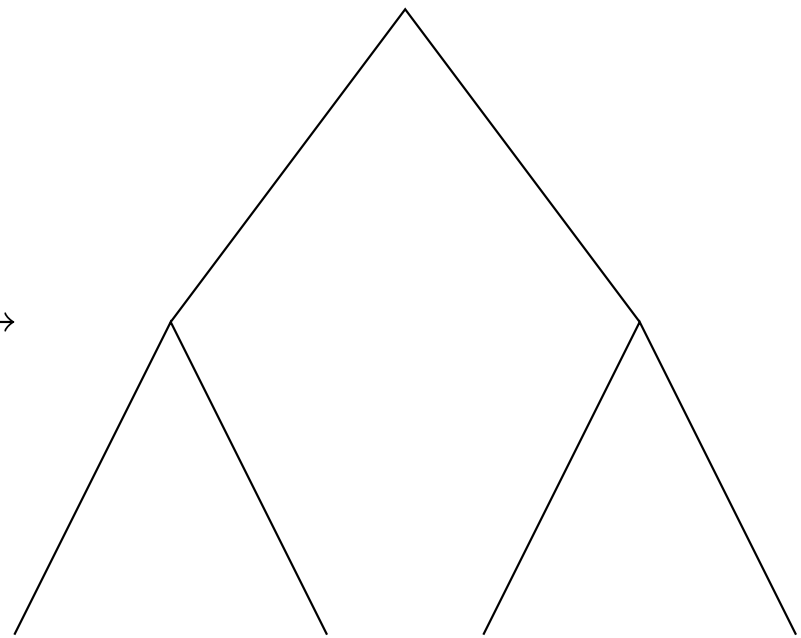


Qu'est-ce qu'un *parser* ?

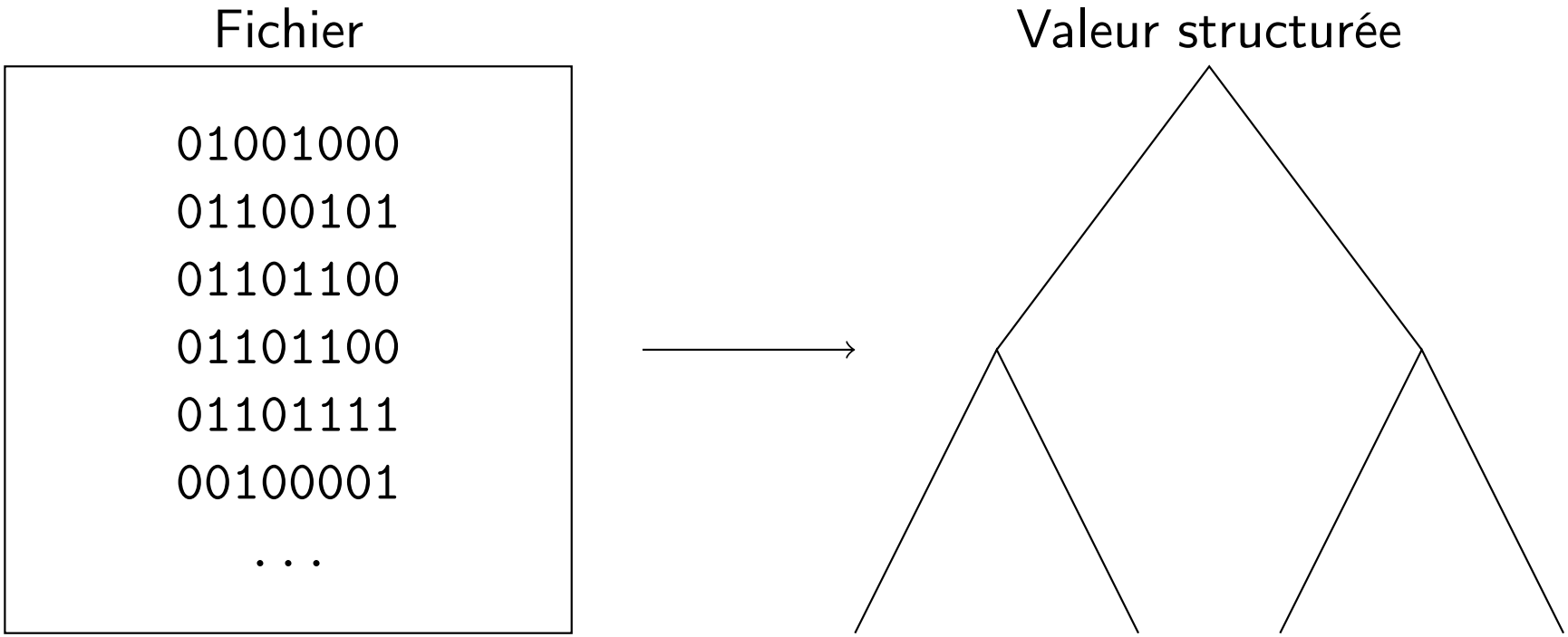
Fichier

```
int main () {
  printf ("...");
  ...
}
```

Valeur structurée



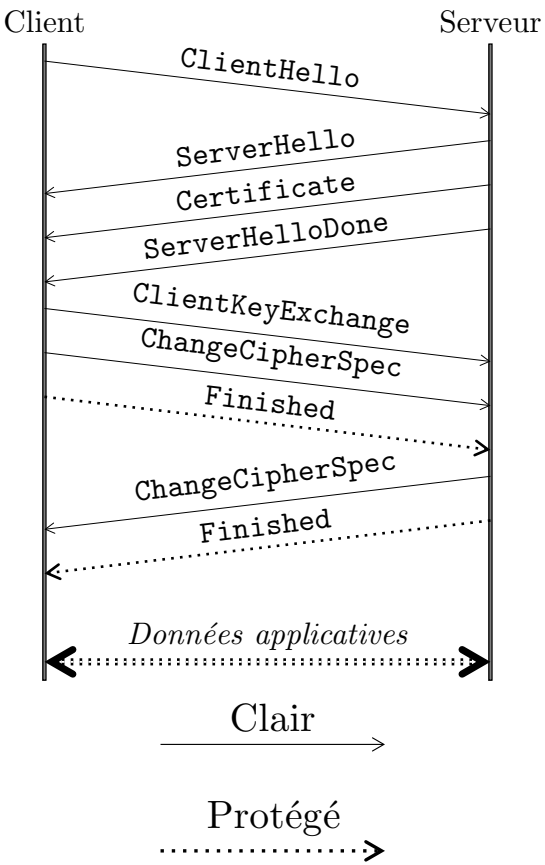
Qu'est-ce qu'un *parser* ?

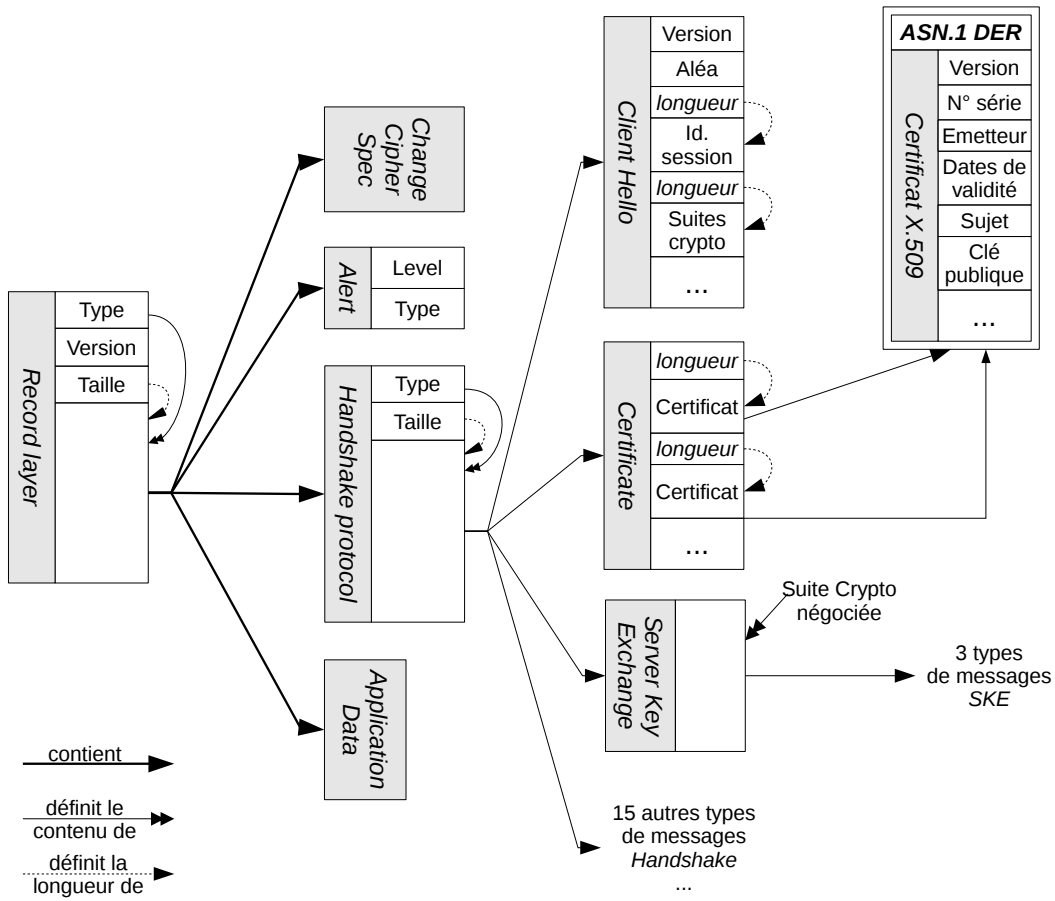


Protocoles réseau et formats de fichiers

- 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, ordre de remplissage d'un octet bit-à-bit
  - spécifications floues
- Les *parsers* binaires sont une brique de base de nombreux programmes
- 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...

SSL/TLS : le point de départ de Parsifal





Analyse de données SSL/TLS (1/2)

- Analyse de mesures SSL/TLS (ACSAC 2012, thèse)
  - pour chaque hôte contacté, réponse du serveur à un stimulus
  - 200 Go de données brutes
- Problème pour disséquer toutes ces données
  - messages structurés complexes
  - données corrompues
  - protocole autre que SSL/TLS (en général HTTP ou SSH)
  - erreurs plus subtiles dans les messages

TLS : la réalité du terrain

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

TLS : la réalité du terrain

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

A `AES128-SHA`

TLS : la réalité du terrain

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

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

TLS : la réalité du terrain

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

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C une alerte

TLS : la réalité du terrain

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

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C une alerte

D la réponse D (`RC4_MD5`)

TLS : la réalité du terrain

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

A `AES128-SHA` (0x002f)

B `ECDH-ECDSA-AES128-SHA` (0xc005)

C une alerte

D la réponse D (`RC4_MD5`) (0x0005)

Le pire, c'est qu'on peut l'expliquer :

- une suite cryptographique est un entier sur 16 bits
- pendant longtemps, les seules valeurs utilisées étaient `00 XX`
- du coup, pourquoi considérer l'octet de poids fort ?

TLS : la réalité du terrain

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

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C une alerte

D la réponse D (`RC4_MD5`)

E un `ServerHello` auquel il *manque* deux octets !

Outils existants pour analyser des formats binaires

Utiliser les bibliothèques standards

- certaines sont fragiles
- incomplètes
- silencieusement laxistes

Regarder les outils disponibles à l'époque

- Scapy / Hachoir
- [+] de nombreux protocoles et formats déjà implémentés
- [+] des logiciels facilement extensibles
- [–] expressivité limitée
- [–] peu de garantie apportée par le langage

Implémentations maison

- Python, C++, OCaml... puis Parsifal (préprocesseur et bib. OCaml)

Motivation initiale

Parsifal

Limitations et nouvelles motivations : vers Parifal v2 ?

Parsifal : plaquette publicitaire

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

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

Parsifal : idée de base

Description des objets à analyser avec des PTypes

- un type OCaml
- une fonction `parse`
- une fonction `dump`

Différentes sortes de PTypes

- les PTypes de base (`uint`, `binstring`, etc.)
- les constructions Parsifal (`enum`, `struct`, etc.)
- les PTypes écrits à la main

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 (2/3)

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

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 (3/3)

```
struct image_header = {
  ...
}

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
```

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

```
struct image_header = {
  ...
}

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

Autres fonctionnalités

Au-delà de `struct` et `union`, Parsifal connaît

- des constructions pour décrire des structures ASN.1 au format DER
- les champs de bits et énumération
- une notion de `containers` servant à
  - la compression (`ztext : zlib_container of string;`)
  - l'encodage (par exemple `base64`)
  - des transformations cryptographiques (ex. : `pkcs1_container`)
- une boîte à outils de PTypes prédéfinis

Autres fonctionnalités

Au-delà de `struct` et `union`, Parsifal connaît

- des constructions pour décrire des structures ASN.1 au format DER
- les champs de bits et énumération
- une notion de `containers` servant à
  - la compression (`ztext : zlib_container of string;`)
  - l'encodage (par exemple `base64`)
  - des transformations cryptographiques (ex. : `pkcs1_container`)
- une boîte à outils de PTypes prédéfinis

Avec ce marteau, tous les formats binaires nous ont paru être des clous...

Parsifal : quelques réalisations

Formats implémentés :

X.509	description assez complète
SSL/TLS	beaucoup de messages décrits implémentation rudimentaire
Kerberos	messages PKINIT
BGP/MRT	extraction des annonces de préfixes
DNS	tutoriel + `picodig`
NTP	quelques messages
TAR	tutoriel
PNG	tutoriel
OpenPGP	structure des paquets
DVI	dissection rudimentaire

Des outils robustes, qui ont servi pour plusieurs publications

Motivation initiale

Parsifal

Limitations et nouvelles motivations : vers Parifal v2 ?

Limitations

Limitations de Parsifal

- adhérence à OCaml...
- et en particulier à `camlp4`
- gestion sordide des formats non linéaires
- absence d'un interpréteur sympa pour explorer à la main

Nouvelles idées

- regarder d'autres langages, comme Rust (et sa bibliothèque Nom)
- enrichir le DSL pour pouvoir raisonner sur les PTypes
- meilleures gestion des contraintes
- meilleure séparation entre *parsing* et sémantique

PDF : exemple d'une spécification intéressante (1/2)

PDF : un format mêlant aspects textuels et binaires

PDF : exemple d'une spécification intéressante (1/2)

PDF : un format mêlant aspects textuels et binaires

Des fonctionnalités complexes

- mises à jour incrémentale
- compression des flux, des objets
- fichiers *linéarisés*
- renseignement tardif de la longueur d'un objet

PDF : exemple d'une spécification intéressante (1/2)

PDF : un format mêlant aspects textuels et binaires

Des fonctionnalités complexes

- mises à jour incrémentale
- compression des flux, des objets
- fichiers *linéarisés*
- renseignement tardif de la longueur d'un objet

Problèmes induits

- lecture non linéaire dans le cas général
- possibilité d'avoir des informations incohérentes

PDF : exemple d'une spécification intéressante (2/2)

> *When a conforming reader reads a PDF file with a damaged or missing cross-reference table, it **may attempt** to rebuild the table by scanning all the objects in the file.*
>
> — *ISO 32000-1-2008, annex C.2*

PDF est un format

- complexe
- mal spécifié
- qui facilite l'*écriture* de fichiers, non sa *lecture*

Pour en savoir plus :

- `https://paperstreet.picty.org/yeye/tag/pdf.html`
- `https://github.com/caradoc-org/caradoc`

Nouvelle vision des `struct`

```
struct png_chunk = {
  chunk_size : uint32;
  chunk_type : string(4);
  chunk_data : chunk_content;
  chunk_crc : uint32;
} constraints {
  chunk_size = len(chunk_content);
  chunk_crc = crc32(chunk_type ^ chunk_data);
  chunk_type = discriminant (chunk_content)
}
```

Nouvelle vision des `struct`

```
struct png_chunk = {
  chunk_size : uint32;
  chunk_type : string(4);
  chunk_data : chunk_content;
  chunk_crc : uint32;
} constraints {
  chunk_size = len(chunk_content);
  chunk_crc = crc32(chunk_type ^ chunk_data);
  chunk_type = discriminant (chunk_content)
}
```

Idées des contraintes :

- définir des relations fonctionnelles pour le *parsing* et le *dumping*
- produire un `png_chunk` valide ne requiert que le champ `data`
  - `chunk_data = ImageHeader ...` implique que
  - `chunk_size` est calculable
  - `chunk_type` type est `"IHDR"`
  - `chunk_crc` est calculable

P2 ?

Projet pour la nouvelle version de Parsifal

- un DSL plus riche pour définir les structures
- un ensemble de PTypes prédéfinis pour un certain nombre de langages
- un compilateur du DSL vers ces langages
- un interpréteur prenant en entrée une spécification et un fichier

P2 ?

Projet pour la nouvelle version de Parsifal

- un DSL plus riche pour définir les structures
- un ensemble de PTypes prédéfinis pour un certain nombre de langages
- un compilateur du DSL vers ces langages
- un interpréteur prenant en entrée une spécification et un fichier

Points d'attention

- définir un langage riche, mais aussi élégant et simple
- intégrer un moyen d'écrire des PTypes à la main
- étudier la manière de *parser* en plusieurs temps

Après P2 ?

Animation de protocole

- un DSL pour décrire l'état interne d'un acteur
- un DSL pour décrire la machine à état à partir des messages, en incluant la manipulation de l'état
- un compilateur produisant des implémentations de référence
- un *fuzzer* de machines à état utilisant ces descriptions
- protocoles visés : TLS, BGP, H2, SSH, QUIC

Après P2 ?

Animation de protocole

- un DSL pour décrire l'état interne d'un acteur
- un DSL pour décrire la machine à état à partir des messages, en incluant la manipulation de l'état
- un compilateur produisant des implémentations de référence
- un *fuzzer* de machines à état utilisant ces descriptions
- protocoles visés : TLS, BGP, H2, SSH, QUIC

Extension aux formats de fichiers

- l'interprétation des *chunks* PNG est décrit par une machine à état
- le traitement d'un fichier DVI

Conclusion et perspectives

Parsifal : un joli marteau qui a bien servi

- méthode de développement éprouvée
- résultats sur plusieurs études de cas
- limitations
  - dues à des choix technologiques
  - dues à l'expressivité limitée du langage de description

Vers une nouvelle version (P2)

- remise à plat du langage
- un compilateur à part entière pour pouvoir cibler différents langages
- réflexion sur la description de protocoles
- projet ANR JCJC soumis en 2018 (GASP)

Questions ?

Merci de votre attention.

olivier.levillain@telecom-sudparis.eu

Code source	https://github.com/picty/parsifal
Publications	https://paperstreet.picty.org/yeye/tag/parsifal.html