Parsifal: a pragmatic solution to the binary parsing problem

Olivier Levillain

ANSSI

May 18th 2014
Motivation: studying SSL/TLS answers

Parsifal

Results

Lessons learned
Agenda

Motivation: studying SSL/TLS answers

Parsifal

Results

Lessons learned
Analysing SSL/TLS data

How to analyse the 180 GB of data collected on port 443?

- complex message format
- presence of corrupted data
- presence of other protocols (HTTP, SSH...)
- more subtle errors may arise
Analysing SSL/TLS data

How to analyse the 180 GB of data collected on port 443?

▶ complex message format
▶ presence of corrupted data
▶ presence of other protocols (HTTP, SSH...)
▶ more subtle errors may arise

What should you expect from a server when you only propose the AES128-SHA and DHE-RSA-AES128-SHA ciphersuites?
Motivation

Analysing SSL/TLS data

How to analyse the 180 GB of data collected on port 443?

▶ complex message format
▶ presence of corrupted data
▶ presence of other protocols (HTTP, SSH...)
▶ more subtle errors may arise

What should you expect from a server when you only propose the AES128-SHA and DHE-RSA-AES128-SHA ciphersuites?

A AES128-SHA

Analysing SSL/TLS data

How to analyse the 180 GB of data collected on port 443?

▶ complex message format
▶ presence of corrupted data
▶ presence of other protocols (HTTP, SSH...)
▶ more subtle errors may arise

What should you expect from a server when you only propose the AES128-SHA and DHE-RSA-AES128-SHA ciphersuites?

A  AES128-SHA
B  DHE-RSA-AES128-SHA
Analysing SSL/TLS data

How to analyse the 180 GB of data collected on port 443?

- complex message format
- presence of corrupted data
- presence of other protocols (HTTP, SSH...)
- more subtle errors may arise

What should you expect from a server when you only propose the **AES128-SHA** and **DHE-RSA-AES128-SHA** ciphersuites?

A  **AES128-SHA**

B  **DHE-RSA-AES128-SHA**

C  an alert
Motivation

Analysing SSL/TLS data

How to analyse the 180 GB of data collected on port 443?

- complex message format
- presence of corrupted data
- presence of other protocols (HTTP, SSH...)
- more subtle errors may arise

What should you expect from a server when you only propose the \texttt{AES128-SHA} and \texttt{DHE-RSA-AES128-SHA} ciphersuites?

A \texttt{AES128-SHA}

B \texttt{DHE-RSA-AES128-SHA}

C an alert

D something else (\texttt{RC4_MD5})
Motivation

Existing tools

To reliably analyse the data, we needed relatively fast and reliable tools

- they should handle gracefully corrupted (or even malicious) input

Standard TLS stacks did not meet our needs, since they can be

- fragile
- incomplete
- silently laxist

Among the existing tools to write parsers, we found nothing suitable:

- Scapy/Hachoir, Python tools
- existing Haskell/OCaml libraries
- binpac, a C preprocessor from the Bro project
Existing tools

To reliably analyse the data, we needed relatively fast and reliable tools

- they should handle gracefully corrupted (or even malicious) input

Standard TLS stacks did not meet our needs, since they can be

- fragile
- incomplete
- silently laxist

Among the existing tools to write parsers, we found nothing suitable:

- Scapy/Hachoir, Python tools
- existing Haskell/OCaml libraries
- binpac, a C preprocessor from the Bro project
- Nail
Homemade tools

To handle the SSL/TLS data, several *parsers* were developed, using different languages

- **Python**: quick to write, but too slow at runtime
- **C++**: flexible, fast at runtime, but verbose and hard to debug
- **OCaml**: robust, efficient, but still too much code
- **OCaml with an integrated preprocessor**: everything looks fine
Agenda

Motivation: studying SSL/TLS answers

Parsifal

Results

Lessons learned
Parsifal lets you describe constructions
The corresponding parsing (and dumping) functions are generated
For example, a simple DNS client can fit in 200 locs
Parsifal lets you describe constructions
The corresponding parsing (and dumping) functions are generated
For example, a simple DNS client can fit in 200 locs

With Parsifal, parsers can be written with **concise** code
The resulting programs are **fast**
They are also **robust**
Parser can be developed **incrementally**
Parsifal lets you describe constructions

The corresponding parsing (and dumping) functions are generated

For example, a simple DNS client can fit in 200 locs

With Parsifal, parsers can be written with **concise** code

The resulting programs are **fast**

They are also **robust**

Parsers can be developed **incrementally**

Possible usages of Parsifal

- robust analysis tools
- basic blocks for sanitisation tools
- secure protocol implementations
First example: a trivial PNG parser

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

First example: a trivial PNG parser

```ocaml
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)
```
First example: a trivial PNG parser

```ocaml
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)
```

Program output:

```ocaml
value {
    png_magic: 89504e470d0a1a0a (8 bytes)
    png_content: 0000000d49484... (264 bytes)
}
```
Chunk handling (1/2)

```c
struct png_file = {
    png_magic : magic("\x89\x50\x4e\x47\x0d\x0a\x1a\x0a"),
    png_content : list of chunk;
};
```
Chunk handling (1/2)

```c
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;
}
```
Chunk handling (2/2)

Sortie du programme:

```plaintext
value {
    png_magic: 89504e470d0a1a0a (8 bytes)
    chunks {
        chunks[0] {
            chunk_size: 13 (0x0000000d)
            chunk_type: "IHDR" (4 bytes)
            data: 000000140000000160403000000 (13 bytes)
            crc: 846176565 (0x326fa135)
        }
        ... 4 other chunks ...
    }
}
```
Chunk enriching: IHDR

```c
struct chunk = {
    chunk_size : uint32;
    chunk_type : string(4);
    data : container(chunk_size) of chunk_content;
    crc : uint32;
}
```
Chunk enriching: IHDR

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

union chunk_content [enrich] (UnparsedChunkContent) =
| "IHDR" → ImageHeader of image_header
```
Chunk enriching: IHDR

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

union chunk_content [enrich] (UnparsedChunkContent) =
| "IHDR" → ImageHeader of image_header

struct image_header = {
    width: uint32; height : uint32;
    bit_depth : uint8;
    color_type : color_type;
    ...
}
```
Chunk enriching: IHDR

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

union chunk_content [enrich] (UnparsedChunkContent) =
| "IHDR"  → ImageHeader of image_header

struct image_header = {
    width: uint32; height : uint32;
    bit_depth : uint8;
    color_type : color_type;
    ...
}

enum color_type (8, UnknownVal UnknownColorType) =
| 0   → Grayscale
| 2   → Truecolor
...```
Features

Beyond `enum`, `struct` and `union`, Parsifal also has

- `asn1_*` keywords to describe ASN.1 structures (DER format)
- bit fields
- a notion of containers to automate:
  - compression (`ztext : zlib_container of string;`)
  - encoding (e.g. `base64`)
  - cryptographic transformations (e.g. `pkcs1_container`)
  - additional constraints
- a toolbox of predefined PTypes

The produced tools are robust against invalid inputs, by construction

- static typing
- strict interpretation
Features

Beyond enum, struct and union, Parsifal also has

- asn1_* keywords to describe ASN.1 structures (DER format)
- bit fields
- a notion of containers to automate:
  - compression (ztext : zlib_container of string;)
  - encoding (e.g. base64)
  - cryptographic transformations (e.g. pkcs1_container)
  - additional constraints
- a toolbox of predefined PTypes

The produced tools are robust against invalid inputs, by construction

- static typing
- strict interpretation

Once we had this hammer, every binary format really looked like a nail
Features

Beyond `enum`, `struct` and `union`, Parsifal also has

- `asn1_*` keywords to describe ASN.1 structures (DER format)
- bit fields
- a notion of containers to automate:
  - compression (`ztext : zlib_container of string;`)
  - encoding (e.g. `base64`)
  - cryptographic transformations (e.g. `pkcs1_container`)
  - additional constraints
- a toolbox of predefined PTypes

The produced tools are robust against invalid inputs, by construction

- static typing
- strict interpretation

Once we had this hammer, every binary format *really* looked like a nail

But Parsifal always allows to mix manually written types
Agenda

Motivation: studying SSL/TLS answers

Parsifal

Results

Lessons learned
Some figures

Three home-made TLS analysers (certificate extraction)

<table>
<thead>
<tr>
<th></th>
<th>C++</th>
<th>OCaml</th>
<th>Parsifal</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>8,500</td>
<td>4,000</td>
<td>1,000</td>
</tr>
<tr>
<td>Processing time</td>
<td>100 s</td>
<td>40 s</td>
<td>8 s</td>
</tr>
</tbody>
</table>

Three tools to analyse BGP messages:

<table>
<thead>
<tr>
<th></th>
<th>libbgpdump</th>
<th>OCaml</th>
<th>Parsifal</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>4,000</td>
<td>1,200</td>
<td>550</td>
</tr>
<tr>
<td>Processing time</td>
<td>23 s</td>
<td>180 s</td>
<td>35 s</td>
</tr>
<tr>
<td>Robustness</td>
<td>NO</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Other formats

Here are a list of formats (at least partially) implemented

▶ DNS
▶ NTP
▶ PNG
▶ OpenPGP
▶ Kerberos
▶ PE
▶ UEFI Firmware
▶ DVI
Agenda

Motivation: studying SSL/TLS answers

Parsifal

Results

Lessons learned
On formats

There is something as a bad format:

- PE and EXIF include non-linear structures
- DVI force you to know the whole spec to parse a file
- integers may come in very different flavours
On formats

There is something as a bad format:

- PE and EXIF include non-linear structures
- DVI force you to know the whole spec to parse a file
- integers may come in very different flavours
  - at least 4 in ASN.1 DER
  - do you know the TAR way to represent them?
Lessons learned

On formats

There is something as a bad format:

- PE and EXIF include non-linear structures
- DVI force you to know the whole spec to parse a file
- integers may come in very different flavours
  - at least 4 in ASN.1 DER
  - do you know the TAR way to represent them?

On the contrary, we like

- Tag-Length-Value which allows extensibility
- canonical representations
- reusable elements
- simple, linear parsing
On the language

- OCaml proved to be a robust language
- The presence of a GC is often seen as a major advantage
- (unless you *want* to handle some memory cells)
- For me, the real pro is the exhaustive pattern matching
- Also, strong typing keep you on track
On the process

- Implementing parsers gives you real insight in formats and protocols
- Parsifal automates most of the mind-numbing repetitive tasks
- Parsifal allows for incremental development
On the process

- Implementing parsers gives you real insight in formats and protocols
- Parsifal automates most of the mind-numbing repetitive tasks
- Parsifal allows for incremental development

- Yet our methodology aims at checking the validity of values with robust tools, not so much at fuzzing
Conclusion

- Three years of writing parsers led us to Parsifal
- Our hammer looks more and more like a Swiss knife
- Until now, we mainly used it to understand formats and analyse data
- Sanitization tools have been prototyped (certificates, PNG)
- Next step: more real-world use cases
- Since June 2013, the code is available on GitHub
Questions?

Thank you for your attention

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

olivier.levillain@ssi.gouv.fr