Tools Techniques & Theory – Security meets Language Theory

Research in Progress

Norina Marie Grosch, Joshua Koenig, Stefan Lucks

Bauhaus-Universität Weimar, Germany

January 19, 2017

Motivating Example: Heartbleed

```
/* Allocate memory for the response,
 * size is 1 bytes message type,
 * plus 2 bytes payload length,
 * plus payload, plus padding
 */
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);

r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT,
                     buffer, 3 + payload + padding);
```

- `payload`, `padding`: lengths (in bytes)
- `bp`: pointer to output buffer
- `pl`: pointer to start of payload in input buffer

Where is the problem? 

- The statement

```
memcpy(bp, pl, payload);
```

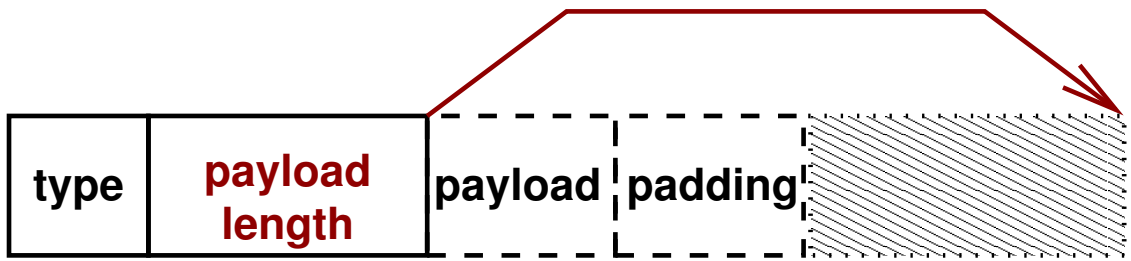
copies `payload` byte from `pl` to `bp`.

Where is the problem?

- The statement

```
memcpy(bp, pl, payload);
```

  copies `payload` byte from `pl` to `bp`.
- If `payload` is larger than the actual length of `pl`, the server will any data to the attacker, which just happen to be stored at that locatiion.

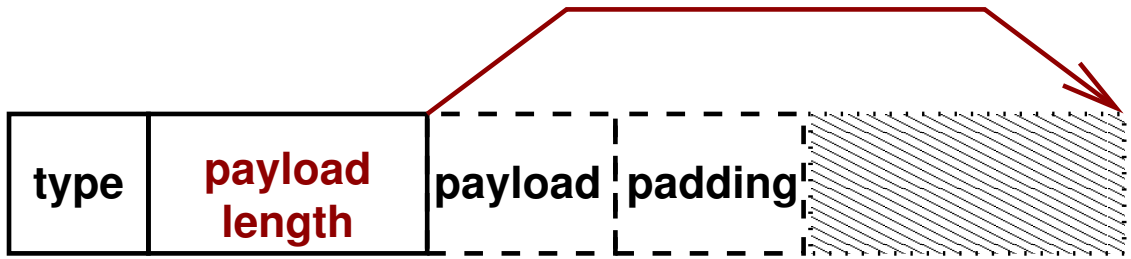


Where is the problem?

- The statement

```
memcpy(bp, pl, payload);
```

  copies `payload` byte from `pl` to `bp`.
- If `payload` is larger than the actual length of `pl`, the server will any data to the attacker, which just happen to be stored at that locatiion.



- The variable `payload` holds the “payload length” (2 bytes). If payload and padding bytes are short, the adversary can receive almost 2^{16} Bytes from the server’s memory.

Why?

Slide from Bratus, Patterson (2012): Shotgun parsers in the cross-hairs
`http://langsec.org/brucon/ShotgunParsersBruCON.pdf`

Shotgun Parsers

- Input data checking, handling interspersed with processing logic

The General Problem!

- Lack of precise specifications.
- Lack of tools to precisely specify binary interfaces.
- Lack of tools to translate the precise specification into code.

The General Problem!

- ! Lack of precise specifications.
- ! Lack of tools to precisely specify binary interfaces.
- ! Lack of tools to translate the precise specification into code.

- ? But we have tools to precisely specify languages (Regular Expressions, Extended Backus-Naur Form, . . .).
- ? And we have tools to generate parsers from specifications (lex, yacc, bison, . . .).
- ? Why don't we use them?

The General Problem!

- ! Lack of precise specifications.
- ! Lack of tools to precisely specify binary interfaces.
- ! Lack of tools to translate the precise specification into code.

- ? But we have tools to precisely specify languages (Regular Expressions, Extended Backus-Naur Form, . . .).
- ? And we have tools to generate parsers from specifications (lex, yacc, bison, . . .).
- ? Why don't we use them?

- ! Because the "tools to precisely specify" don't apply to "length-prefix" languages, such as the TLS binary iterface. See below!

What, e.g., about ASN.1?

- TLS uses the “transfer syntax” from ASN.1.
- The bug is about properly parsing this very “transfer syntax”.
- The “transfer syntax” specification (and the “encoding rules”) are English prose.

What, e.g., about ASN.1?

- TLS uses the “transfer syntax” from ASN.1.
- The bug is about properly parsing this very “transfer syntax”.
- The “transfer syntax” specification (and the “encoding rules”) are English prose.

  Nothing wrong with English prose.

  It may even win you the Nobel prize for Literature . . .

What, e.g., about ASN.1?

- TLS uses the “transfer syntax” from ASN.1.
- The bug is about properly parsing this very “transfer syntax”.
- The “transfer syntax” specification (and the “encoding rules”) are English prose.

  Nothing wrong with English prose.

  It may even win you the Nobel prize for Literature . . .

Other binary interfaces with length-prefix or count-prefix notation: Apache Avro, Bencode, Binn, BSON, . . .

Also: Almost all file formats for sound, pictures and movies (PNG ect.).

Formal Language Theory

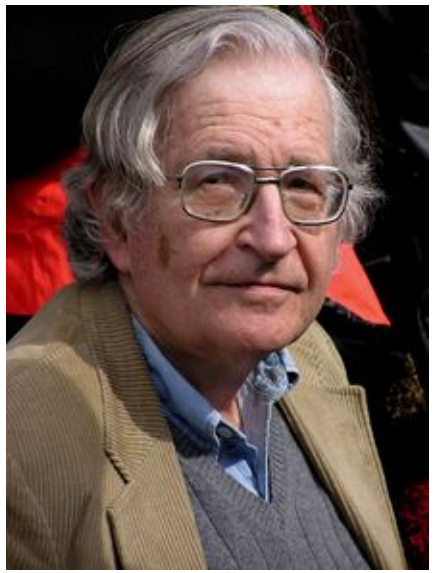
- “old and dusted”
- practical problems “solved” since the late 1970’s
- now the theory strikes back . . . languages are everywhere:
  - network stacks: valid packets make a language (stack is a recognizer at every layer/protocol)
  - servers: valid requests make a language (E.g., SQL injection as a recognizer failure)
  - memory management: heaps and stacks make a language
    (memory metadata is even context-sensitive)

  . . . and these languages are often ambigously specified and implemented by unsystematic ad-hoc parsers, who’s authors apparently did not consider their program as a “parser”

Chomsky Hierarchy

Noam Chomsky, “Three Models for the Description of Language” (1958)

- type 0: recursively enumerable
- type 1: context-sensitive
- type 2: context-free
  - full context-free (recognize by non-deterministic push-down automatons)
  - weakly context-free (recognize by deterministic push-down automatons)
- type 3: regular (recognize by finite automatons)



Noam Chomsky

Worst-Case Feasibility

	"word problem" is w in L?	"computational equivalence"
type 0:	undecidable	undecidable
type 1: context-sensitive	**exponential**	undecidable
type 2: context-free	cubic	undedicable
type 2: det. context-free	linear	decidable
type 3: regular	linear	decidable
type 3:	$O(1)$ storage	

computational equivalence:

do "they" generate/accept the same language?

Concrete Example: Netstrings

Bernstein (1999)

- Number N (in decimal)
- Colon (“:”)
- N-byte String
- Comma (“,”)

“Hello World!” = 12:Hello World,

Concrete Example: Netstrings

Bernstein (1999)

- Number N (in decimal)
- Colon (":")
- N-byte String
- Comma (",")

"Hello World!" = 12:Hello World,

Nested Netstrings
("heartbeat-like" with challenge abc and Padding XY):

8:3:abc,XY,

Heartbleed-like **attack package:**

5:9999:,

What is “wrong” with Netstrings?

and other length-prefix languages

Pumping Lemma

If L is an infinite language over σ accepted by a PDA, then there exist words $u, v, w, x, y \in \Sigma^*$ with $v \neq \epsilon$ and $x \neq \epsilon$ such that

$$uv^i wx^i y \in L$$

for all $i \geq 0$

Theorem

Netstrings are not context-free. (Even without nesting.)

In practice, this means we cannot parse Netstrings (and other length-prefix languages) using the tools from Formal Language Theory. **So we need new tools!** (Or we should abandon length-prefix languages for binary interfaces.)

Calc-Regular Expressions

- Regular Expressions + two additional kinds of expressions

Calc-Regular Expressions

- Regular Expressions + two additional kinds of expressions

1. Expressions for length-prefixes:

$$r_1 = x(u.f)y(v\#f)w$$

A word from $L(r_1)$ is the concatenation of

1.1. a word from $L(x)$,
1.2. a word w_u from $L(u)$,
1.3. a word from $L(y)$,
1.4. a word from $L(v)$) of exactly $f(w_u)$ bytes, and
1.5. a word from $L(z)$.

Calc-Regular Expressions

- Regular Expressions + two additional kinds of expressions

1. Expressions for length-prefixes:

$$r_1 = x(u.f)y(v\#f)w$$

A word from $L(r_1)$ is the concatenation of

1.1. a word from $L(x)$,
1.2. a word w_u from $L(u)$,
1.3. a word from $L(y)$,
1.4. a word from $L(v)$) of exactly $f(w_u)$ bytes, and
1.5. a word from $L(z)$.

2. Expressions for calc-prefixes.

$$r_2 = x(u.f)y(v^f)w$$

a word from $L(r_2)$ is the concatenation of ...

2.4. exactly $f(2_u)$ words from $L(v)$), and ...

More work to do!

We are just Beginning

- Theoretical Properties of Calc-Regular languages (closed under intersection, not closed under union, . . .)
- Parser generator (similar to lex)
- Calc-Context-Free languages
- . . .

Security and Formal Languages?

use the right tool for the job at hand!

- Security is (mostly) about attacking the interface between two communicating entities.
- The entities use some language to communicate.
- Formal Language Theory is about
  - understanding languages,
  - properly un-ambigously defining them,
  - properly parsing them,
  - ...

This field of research has been pioneered by Sergey Bratus, Meredith L. Patterson, and Len Sassaman

`http://www.cs.dartmouth.edu/~{}sergey/`.

Fun Fact: PDF Specs allow Unparsable Objects

Bogk, Schöpfl, 2014

Recursion:
The length of object 6
depends on the content of
object 6.
Before parsing object 6,
you must parse object 6.

```
6 0 obj
<< /Length 6 0 R >>
stream
foobar
endstream
endobj
```

Fun Fact: PDF Specs allow Unparsable Objects

Bogk, Schöpfl, 2014

Recursion:

The length of object 6 depends on the content of object 6.

Before parsing object 6, you must parse object 6.

```
6 0 obj
<< /Length 6 0 R >>
stream
foobar
endstream
endobj
```

Mutual recursion:

```
7 0 obj
<< /Length 8 0 R >>
stream
foobar
endstream
endobj
8 0 obj
<< /Length 7 0 R >>
stream
foobar
endstream
endobj
```