Network Security

Chapter 2 – Language-theoretic Security

Cornelius Diekmann

Lehrstuhl für Netzarchitekturen und Netzdienste
Institut für Informatik
Technische Universität München

Version: August 31, 2015

Communications protocol

- Defines the procedure and the format of exchanged messages
- Examples
  - IP
  - TCP
  - UDP
  - HTTP
  - HTTPS
  - SSH
  - . . .
- Alice and Bob might speak the same protocol ...
- but do they also have the same understanding?

Problem 1

Example: the X.509 NULL Character “issue”

- Assume you own `zombo.com`
- Then, all subdomains `*.zombo.com` also belong to you
- And you can buy certificates for them

Example: the X.509 NULL Character “issue”

- Assume you own `zombo.com`
- Then, all subdomains `*.zombo.com` also belong to you
- And you can buy certificates for them
- What about

`www.paypal.com\0www.zombo.com`

- where `\0` is the C string terminator (NULL character)
- If a browser accidentally uses `strncmp` to validate certificates ...

Example: the X.509 NULL Character “issue”

- Assume you own `zombo.com`
- Then, all subdomains `*.zombo.com` also belong to you
- And you can buy certificates for them
- What about

```
www.paypal.com\0www.zombo.com
```

- where `\0` is the C string terminator (NULL character)
- If a browser accidentally uses `strncmp` to validate certificates . . .
- . . . you just got a certificate for `www.paypal.com`

Example: the X.509 NULL Character “issue”

- Alice and Bob spoke the same “protocol”: X.509
- But had a different understanding!
- Alice certified the URL: `www.paypal.com\0www.zombo.com`
- Bob parsed the URL: `www.paypal.com`

Problem (1)

- Coder’s implicit assumption

  Input is well-formed

- Reality

  Input is controlled by attacker

Solution (1)

- Apply *full* recognition to inputs before processing them!
- Do not scatter recognition throughout your code!

Problem 2

Example: Recognizing Valid Inputs

- My favorite RFC

```
 Content-Length = 1*DIGIT
[...]
Any Content-Length field value greater than or equal to
zero is valid.  Since there is no predefined limit to the
length of a payload, a recipient MUST anticipate
potentially large decimal numerals and prevent parsing
errors due to integer conversion overflows
```

- Quiz: Which RFC is this taken from?

Example: Recognizing Valid Inputs

- My favorite RFC

```
 Content-Length = 1*DIGIT
[...]
Any Content-Length field value greater than or equal to
zero is valid.  Since there is no predefined limit to the
length of a payload, a recipient MUST anticipate
potentially large decimal numerals and prevent parsing
errors due to integer conversion overflows
```

- Quiz: Which RFC is this taken from?
  - 7230, HTTP/1.1 Message Syntax and Routing

Example: Recognizing Valid Inputs

- My favorite RFC

```
 Content-Length = 1*DIGIT
[...]
Any Content-Length field value greater than or equal to
zero is valid.  Since there is no predefined limit to the
length of a payload, a recipient MUST anticipate
potentially large decimal numerals and prevent parsing
errors due to integer conversion overflows
```

- Quiz: Which RFC is this taken from?
  - 7230, HTTP/1.1 Message Syntax and Routing
- Translation:
  - The length of the content can be arbitrary
  - The length of the `Content-Length` field can be arbitrary
  - Just parse it right

Example: Recognizing Valid Inputs

- What type of grammar is HTTP?
- In the Chomsky hierarchy, at least type 1 – context-sensitive

Example: Recognizing Valid Inputs

- What type of grammar is HTTP?
- In the Chomsky hierarchy, at least type 1 – context-sensitive
- Are two HTTP parsers equivalent?

UNDECIDABLE

Recap (Theoretical Comp. Sci.): Chomsky Hierarchy

Grammar	Language	Recognized by
Type 3	Regular	Finite state automaton
Type 2	Context-free	Pushdown automaton
Type 1	Context-sensitive	Some weird stuff
Type 0	recursively enumerable	Turing machine

$$\text{Type 3} \subset \text{Type 2} \subset \text{Type 1} \subset \text{Type 0}$$

Recap (Theoretical Comp. Sci.): Chomsky Hierarchy

Grammar	Language	Recognized by
Type 3	Regular	Finite state automaton
Type 2	Context-free	Pushdown automaton
Type 1	Context-sensitive	Some weird stuff
Type 0	recursively enumerable	Turing machine

$$\text{Type 3} \subset \text{Type 2} \subset \text{Type 1} \subset \text{Type 0}$$

- Remember all those *undecidable* problems in theo. comp. sci.?

Recap (Theoretical Comp. Sci.): Chomsky Hierarchy

Grammar	Language	Recognized by
Type 3	Regular	Finite state automaton
Type 2	Context-free	Pushdown automaton
Type 1	Context-sensitive	Some weird stuff
Type 0	recursively enumerable	Turing machine

$$\text{Type 3} \subset \text{Type 2} \subset \text{Type 1} \subset \text{Type 0}$$

- Remember all those *undecidable* problems in theo. comp. sci.?
- If the grammar of your protocol is Type 1 or Type 0, you will run into them!

Solution (2)

- Don't define Turing-complete protocols
  - Recognizing is undecidable
  - Testing equivalence of different implementations is undecidable
- With `Content-Length` fields, you easily run into this problem!

Problem 3

Example: Unintended Survey of Visited Porn Pages

- You are visiting my website

Example: Unintended Survey of Visited Porn Pages

- You are visiting my website
- I host a hidden list of links to the most common porn sites

Example: Unintended Survey of Visited Porn Pages

- You are visiting my website
- I host a hidden list of links to the most common porn sites
- Your browser renders
  - Not visited: blue
  - Visited: purple

Example: Unintended Survey of Visited Porn Pages

- You are visiting my website
- I host a hidden list of links to the most common porn sites
- Your browser renders
  - Not visited: blue
  - Visited: purple
- Using JavaScript, the color of the links is send back to me

Solution (3)

- Reduce computing power
- Power that is not there cannot be exploited
- In particular in input handling code

More on Problem (3): “Weird Machines”

- Complex protocols require complex parsers
- Complex parsers (anything beyond Type 2 and 3) expose almost unlimited computational power to the attacker
- Which leads to *“weird machines”*
- A weird machine is a machine programmable by an attacker
- Which was not intended or expected by the programmer

Solution (3) part 2

- Make your protocol context-free or regular
- And use an appropriate parser
  - Parser generators, parser combinators, ...
  - `import re` is not an acceptable solution

Problem 4

Example: Ponies vs. Cats

```
https://www.google.de/webhp?ie=UTF-8&q=ponies&q=cats
```

Example: Ponies vs. Cats

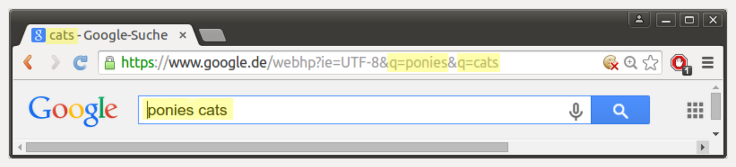
```
https://www.google.de/webhp?ie=UTF-8&q=ponies&q=cats
```

- Alice: “*The user asked for ponies*”
- Bob: “*The user asked for cats*”

Example: Ponies vs. Cats

`https://www.google.de/webhp?ie=UTF-8&q=ponies&q=cats`

- Alice: “*The user asked for ponies*”
- Bob: “*The user asked for cats*”
- Google: “*Let’s go for both (cats preferred)*”



cats - Google-Suche

https://www.google.de/webhp?ie=UTF-8&q=ponies&q=cats

ponies cats

Problem: Mutual Understanding

- Entities may have a different understand of the meaning of a protocol
- In the example
  - Alice recognized the first `q` parameter
  - Bob recognized the last `q` parameter

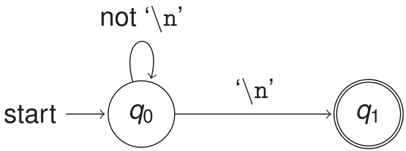
Solution (4)

- Messages must be interpreted the same by all participants
- Parsers must be equivalent
- Only decidable for regular and context-free languages

Examples

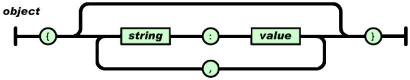
Newline-Delimited

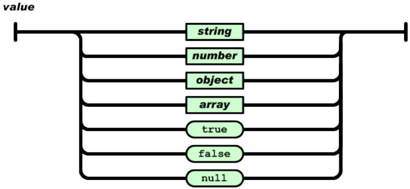
- Familiar from exercises
- Every message is delimited by a '`\n`'
- Nice library support: `sf.readline()`
- Language is *Regular* (Type 3)

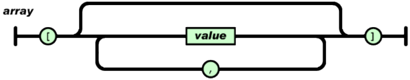


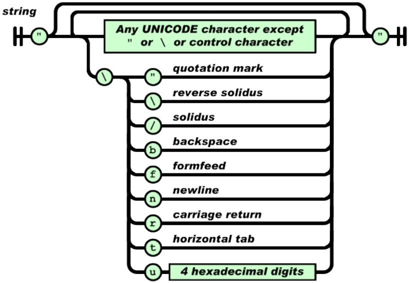
JSON

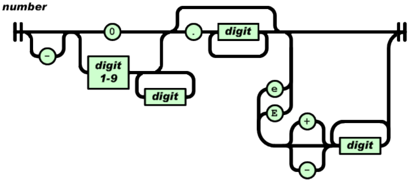
- Context Free (Type 2)







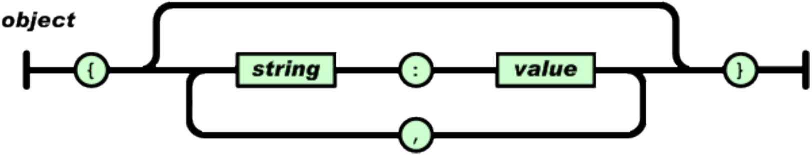




src: json.org

JSON

- Context Free (Type 2)



- But: If unique keys are required → no longer context-free

Literature and Sources

- Len Sassaman, Meredith L. Patterson, Sergey Bratus, Michael E. Locasto, Anna Shubina, *Security Applications of Formal Language Theory*, 2013, `http://langsec.org/papers/langsec-tr.pdf`
- `http://langsec.org/`
- Photoshopped protest signs by Kythera of Anevern (`www.anevern.com`)