The Halting Problems of Network Stack Insecurity

Original paper[1] by Len Sassaman, Meredith L. Patterson, Sergey Bratus and Anna Shubina

Pierre Pavlidès

University of Birmingham - School of Computer Science
Tom Chothia's Internet Security Seminar module

15 March 2013

[1] ;login: The USENIX Magazine, vol. 36, no. 6, December 2011 https://www.usenix.org/publications/login/december-2011-volume-36-number-6/halting-problems-network-stack-insecurity

Before we start

These slides can be downloaded at the following address:

http://r.rogdham.net/17

The full description of the images (including authors and license) is available at the end of this presentation. Indeed, in most of the cases, putting the description next to the image would be given the answer to the questions I may ask during the presentation.

Agenda

1. Language theory in a nutshell
2. Model of the network stack... getting weird
3. Principles of secure design
4. Conclusion

Agenda

1. Language theory in a nutshell
2. Model of the network stack... getting weird
3. Principles of secure design
4. Conclusion

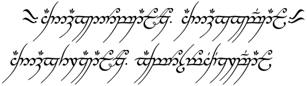
What is a language?

Natural languages

- English
- Sign languages

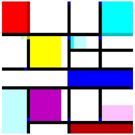
Constructed languages

- Esperanto
- Tolkien's Elvish languages



Programming languages

- C, Python, Java, Haskell, Piet



What about encodings? Data formats?

- HTML, Base64, JSON, PNG

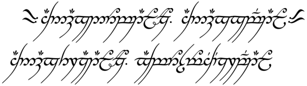
What is a language?

Natural languages

- English
- Sign languages

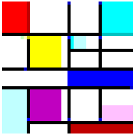
Constructed languages

- Esperanto
- Tolkien's Elvish languages



Programming languages

- C, Python, Java, Haskell, Piet



What about encodings? Data formats?

- HTML, Base64, JSON, PNG

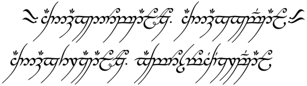
What is a language?

Natural languages

- English
- Sign languages

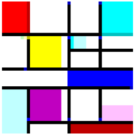
Constructed languages

- Esperanto
- Tolkien's Elvish languages



Programming languages

- C, Python, Java, Haskell, Piet



What about encodings? Data formats?

- HTML, Base64, JSON, PNG

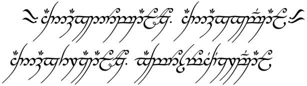
What is a language?

Natural languages

- English
- Sign languages

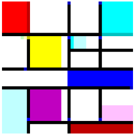
Constructed languages

- Esperanto
- Tolkien's Elvish languages



Programming languages

- C, Python, Java, Haskell, Piet



What about encodings? Data formats?

- HTML, Base64, JSON, PNG

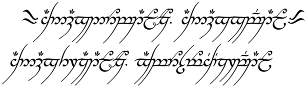
What is a language?

Natural languages

- English
- Sign languages

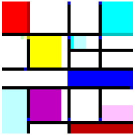
Constructed languages

- Esperanto
- Tolkien's Elvish languages



Programming languages

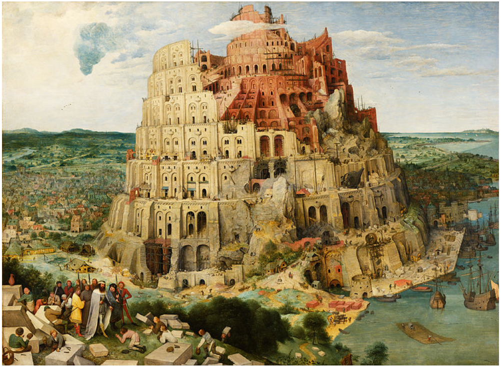
- C, Python, Java, Haskell, Piet



What about encodings? Data formats?

- HTML, Base64, JSON, PNG

What can possibly go wrong?



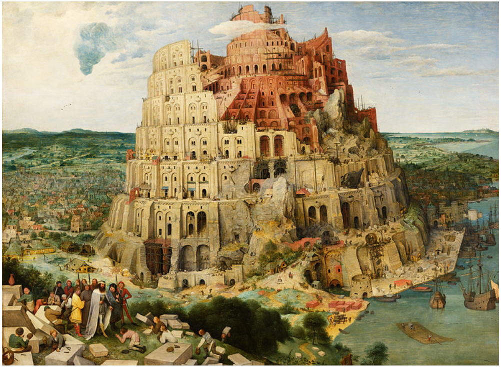
Dialects

- English, Sign languages
- ANSI C: C89. . . C11

Ambiguities

- “not bad”
- “ice cream” / “I scream”

What can possibly go wrong?



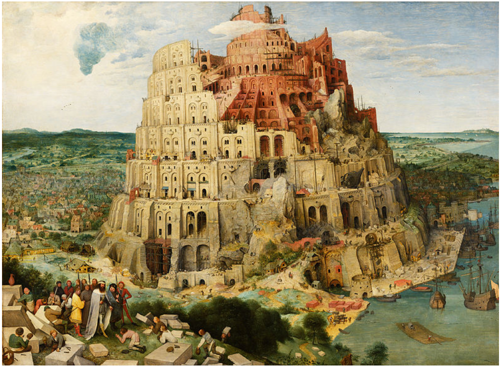
Dialects

- English, Sign languages
- ANSI C: C89. . . C11

Ambiguities

- “not bad”
- “ice cream” / “I scream”

What can possibly go wrong?



Dialects

- English, Sign languages
- ANSI C: C89. . . C11

Ambiguities

- “not bad”
- “ice cream” / “I scream”

Moving to language theory

We need a formal description of a language (read “mathematics”).

And then tools to work on languages:

- recogniser
  - does a word belong to a language?
  - this is an equivalent way to describe a language
- parsers
  - analysing a word of the language, extracting some meaning
  - usually easy once you have the recogniser

Any example of recogniser or parser?

Moving to language theory

We need a formal description of a language (read “mathematics”).

And then tools to work on languages:

- recogniser
    - does a word belong to a language?
    - this is an equivalent way to describe a language
- parsers
    - analysing a word of the language, extracting some meaning
    - usually easy once you have the recogniser

Any example of recogniser or parser?

Moving to language theory

We need a formal description of a language (read “mathematics”).

And then tools to work on languages:

- recogniser
  - does a word belong to a language?
  - this is an equivalent way to describe a language
- parsers
  - analysing a word of the language, extracting some meaning
  - usually easy once you have the recogniser

Any example of recogniser or parser?

Moving to language theory

We need a formal description of a language (read “mathematics”).

And then tools to work on languages:

- recogniser
  - does a word belong to a language?
  - this is an equivalent way to describe a language

- parsers
  - analysing a word of the language, extracting some meaning
  - usually easy once you have the recogniser

Any example of recogniser or parser?

Moving to language theory

We need a formal description of a language (read “mathematics”).

And then tools to work on languages:

- recogniser
  - does a word belong to a language?
  - this is an equivalent way to describe a language
- parsers
  - analysing a word of the language, extracting some meaning
  - usually easy once you have the recogniser

Any example of recogniser or parser?

Recognizer and parser example: email addresses

Recogniser: REGEX

```
if(preg_match(
        '/^[a-z0-9_.-]+@[a-z0-9.-]+\.[a-z0-9-]{2,4}$/',
        $email)) {
    echo 'Valid email address.';
}
```

Parser: just add capturing groups!

```
m = re.match(
    r'^(?P<user>[a-z0-9_.-]+)@'
    r'(?P<domain>[a-z0-9.-]+\.[a-z0-9-]{2,4})$',
    email)
if m and m.group('domain') == 'bham.ac.uk':
    print 'Hello, %s!' % m.group('user')
```

Note: over-simplified REGEX here, do not use it in real life!

Recognizer and parser example: email addresses

Recogniser: REGEX

```
if(preg_match(
        '/^[a-z0-9_.-]+@[a-z0-9.-]+\.[a-z0-9-]{2,4}$/',
        $email)) {
    echo 'Valid email address.';
}
```

Parser: just add capturing groups!

```
m = re.match(
    r'^(?P<user>[a-z0-9_.-]+)@'
    r'(?P<domain>[a-z0-9.-]+\.[a-z0-9-]{2,4})$',
    email)
if m and m.group('domain') == 'bham.ac.uk':
    print  'Hello, %s!' % m.group('user')
```

Note: over-simplified REGEX here, do not use it in real life!

Recognizer and parser example: email addresses

Recogniser: REGEX

```
if(preg_match(
        '/^[a-z0-9_.-]+@[a-z0-9.-]+\.[a-z0-9-]{2,4}$/',
        $email)) {
    echo 'Valid email address.';
}
```

Parser: just add capturing groups!

```
m = re.match(
    r'^(?P<user>[a-z0-9_.-]+)@'
    r'(?P<domain>[a-z0-9.-]+\.[a-z0-9-]{2,4})$',
    email)
if m and m.group('domain') == 'bham.ac.uk':
    print  'Hello, %s!' % m.group('user')
```

Note: over-simplified REGEX here, do not use it in real life!

Some formalism

Definition (Alphabet)

An *alphabet* is a finite set Σ of symbols.

Definition (Word)

A *word* is a finite sequence α of symbols over an alphabet Σ ($\alpha \in \Sigma^*$).

A word can contain spaces!
Notation for the empty word: ϵ.

Definition (Language)

A *language* is a set L of words over an alphabet Σ (i.e. $L \subseteq \Sigma^*$).

A language can contain an infinite number of words (e.g. Σ^*).

Some formalism

Definition (Alphabet)

An *alphabet* is a finite set Σ of symbols.

Definition (Word)

A *word* is a finite sequence α of symbols over an alphabet Σ ($\alpha \in \Sigma^*$).

A word can contain spaces!
Notation for the empty word: ϵ.

Definition (Language)

A *language* is a set L of words over an alphabet Σ (i.e. $L \subseteq \Sigma^*$).

A language can contain an infinite number of words (e.g. Σ^*).

Some formalism

Definition (Alphabet)

An *alphabet* is a finite set Σ of symbols.

Definition (Word)

A *word* is a finite sequence α of symbols over an alphabet Σ ($\alpha \in \Sigma^*$).

A word can contain spaces!
Notation for the empty word: ϵ.

Definition (Language)

A *language* is a set L of words over an alphabet Σ (i.e. $L \subseteq \Sigma^*$).

A language can contain an infinite number of words (e.g. Σ^*).

Some formalism

Definition (Alphabet)

An *alphabet* is a finite set Σ of symbols.

Definition (Word)

A *word* is a finite sequence α of symbols over an alphabet Σ ($\alpha \in \Sigma^*$).

A word can contain spaces!
Notation for the empty word: ϵ.

Definition (Language)

A *language* is a set L of words over an alphabet Σ (i.e. $L \subseteq \Sigma^*$).

A language can contain an infinite number of words (e.g. Σ^*).

Some formalism

Definition (Alphabet)

An *alphabet* is a finite set Σ of symbols.

Definition (Word)

A *word* is a finite sequence α of symbols over an alphabet Σ ($\alpha \in \Sigma^*$).

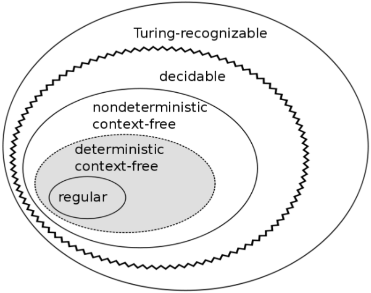
A word can contain spaces!
Notation for the empty word: ϵ.

Definition (Language)

A *language* is a set L of words over an alphabet Σ (i.e. $L \subseteq \Sigma^*$).

A language can contain an infinite number of words (e.g. Σ^*).

Chomsky hierarchy

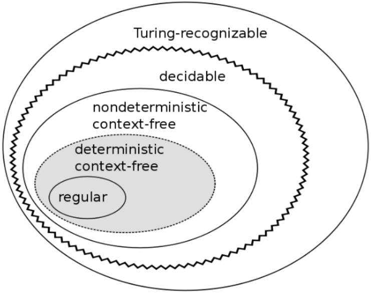


Chomsky hierarchy

Classes of languages

- Regular to recursively enumerable
- How easy it is to recognise a word
- How expressive you can be
- This paper: how secure your application would be

Chomsky hierarchy

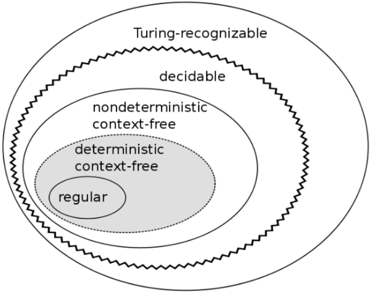


Chomsky hierarchy

Classes of languages

- Regular to recursively enumerable
- How easy it is to recognise a word
- How expressive you can be
- This paper: how secure your application would be

Chomsky hierarchy



Chomsky hierarchy

Classes of languages

- Regular to recursively enumerable
- How easy it is to recognise a word
- How expressive you can be
- This paper: how secure your application would be

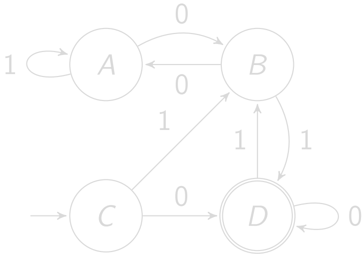
Regular

Summary:

- REGEX
- (Deterministic) finite state automaton

Example:

- multiples of 3 written in binary
- REGEX: `/^(0|(1(01*0)*1))+$/`
- DFSA:



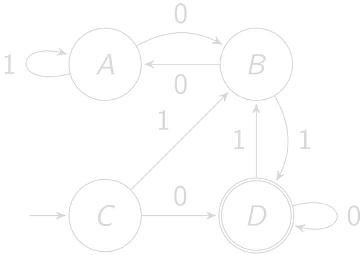
Regular

Summary:

- REGEX
- (Deterministic) finite state automaton

Example:

- multiples of 3 written in binary
- REGEX: `/^(0|(1(01*0)*1))+$/`
- DFSA:



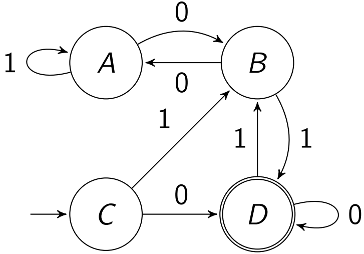
Regular

Summary:

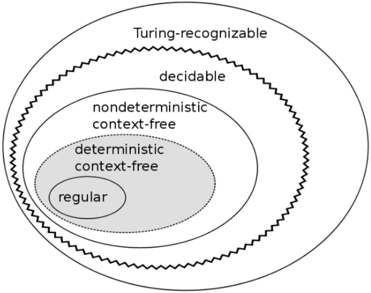
- REGEX
- (Deterministic) finite state automaton

Example:

- multiples of 3 written in binary
- REGEX: `/^(0|(1(01*0)*1))+$/`
- DFSA:



Limit of regular languages



Chomsky hierarchy

Many languages are not regular

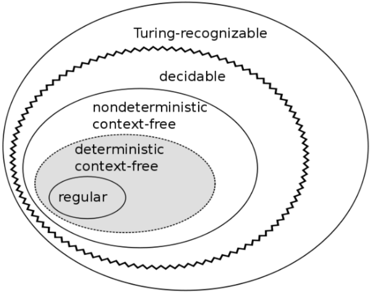
- $L = \{\alpha^n\beta^n | n \in \mathbb{N}\}$
- *Do you think HTML is regular?*

We have other categories of languages!

- need more *context*

I will go directly to recursively enumerable languages.

Limit of regular languages



Chomsky hierarchy

Many languages are not regular

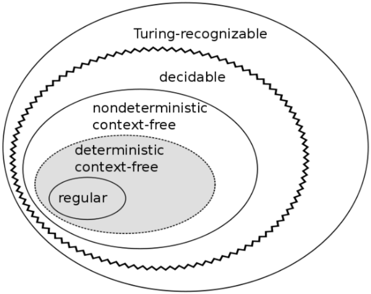
- $L = \{\alpha^n \beta^n | n \in \mathbb{N}\}$
- *Do you think HTML is regular?*

We have other categories of languages!

- need more *context*

I will go directly to recursively enumerable languages.

Limit of regular languages



Chomsky hierarchy

Many languages are not regular

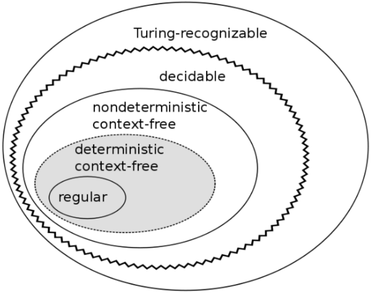
- $L = \{\alpha^n \beta^n | n \in \mathbb{N}\}$
- *Do you think HTML is regular?*

We have other categories of languages!

- need more *context*

I will go directly to recursively enumerable languages.

Limit of regular languages



Chomsky hierarchy

Many languages are not regular

- $L = \{\alpha^n \beta^n | n \in \mathbb{N}\}$
- *Do you think HTML is regular?*

We have other categories of languages!

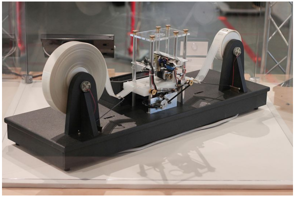
- need more *context*

I will go directly to recursively enumerable languages.

Recursively enumerable languages

Turing machine

- very simple computer; but infinite storage
- tape, head, state register, action table



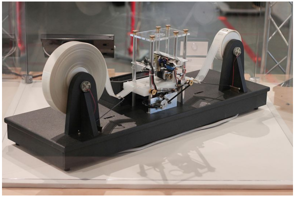
Recursively enumerable languages

- there exists a Turing machine, which, given a word, will
  - halt and accept if part of the language
  - either halt and reject or never halt otherwise
- there exists a Turing machine which will enumerate all the words of the language

Recursively enumerable languages

Turing machine

- very simple computer; but infinite storage
- tape, head, state register, action table



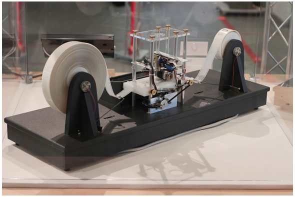
Recursively enumerable languages

- there exists a Turing machine, which, given a word, will
  - halt and accept if part of the language
  - either halt and reject or never halt otherwise
- there exists a Turing machine which will enumerate all the words of the language

Recursively enumerable languages

Turing machine

- very simple computer; but infinite storage
- tape, head, state register, action table



Recursively enumerable languages

- there exists a Turing machine, which, given a word, will
  - halt and accept if part of the language
  - either halt and reject or never halt otherwise
- there exists a Turing machine which will enumerate all the words of the language

Halting problem

YOU ARE NOW INSIDE A MATHEMATICAL SINGULARITY WHERE INFINITY ENDS.
WOW. REALLY?
HERE, THE KOCH SNOWFLAKE IS EVERYWHERE DIFFERENTIABLE, THERE IS NO HALTING PROBLEM, AND AMERICANS HAVE ADOPTED THE METRIC SYSTEM.
YOU'RE KIDDING!
THIS PLACE IS AMAZING.

Given a Turing machine and an input, tell if the machine will eventually halt if run with that input

- undecidable
- basically, you have no choice but to run the machine
  - wait some time
  - if the machine halts, fine
  - else?

Halting problem

YOU ARE NOW INSIDE A MATHEMATICAL SINGULARITY WHERE INFINITY ENDS.
WOW. REALLY?
HERE, THE KOCH SNOWFLAKE IS EVERYWHERE DIFFERENTIABLE, THERE IS NO HALTING PROBLEM, AND AMERICANS HAVE ADOPTED THE METRIC SYSTEM.
YOU'RE KIDDING!
THIS PLACE IS AMAZING.

Given a Turing machine and an input, tell if the machine will eventually halt if run with that input

- undecidable

- basically, you have no choice but to run the machine
  - wait some time
  - if the machine halts, fine
  - else?

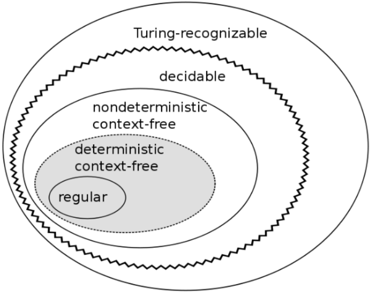
Halting problem

YOU ARE NOW INSIDE A MATHEMATICAL SINGULARITY WHERE INFINITY ENDS.
WOW. REALLY?
HERE, THE KOCH SNOWFLAKE IS EVERYWHERE DIFFERENTIABLE, THERE IS NO HALTING PROBLEM, AND AMERICANS HAVE ADOPTED THE METRIC SYSTEM.
YOU'RE KIDDING!
THIS PLACE IS AMAZING.

Given a Turing machine and an input, tell if the machine will eventually halt if run with that input

- undecidable
- basically, you have no choice but to run the machine
  - wait some time
  - if the machine halts, fine
  - else?

Chomsky hierarchy again



Chomsky hierarchy

Each class of language is included in the next one

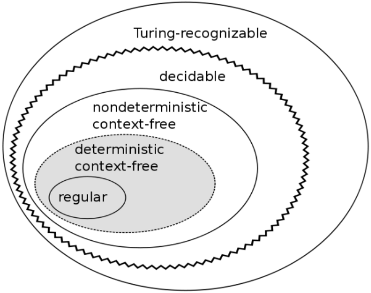
- can handle new languages
- needs more computational power

Grey region: equivalence between two machines is decidable

For each class of language there is a machine which required just the computational power needed

Which one would be more useful to an attacker?

Chomsky hierarchy again



Chomsky hierarchy

Each class of language is included in the next one

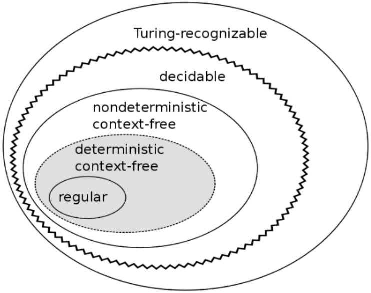
- can handle new languages
- needs more computational power

Grey region: equivalence between two machines is decidable

For each class of language there is a machine which required just the computational power needed

Which one would be more useful to an attacker?

Chomsky hierarchy again



Chomsky hierarchy

Each class of language is included in the next one

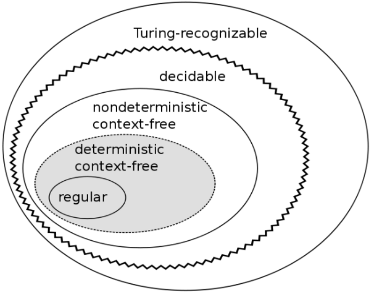
- can handle new languages
- needs more computational power

Grey region: equivalence between two machines is decidable

For each class of language there is a machine which required just the computational power needed

Which one would be more useful to an attacker?

Chomsky hierarchy again



Chomsky hierarchy

Each class of language is included in the next one

- can handle new languages
- needs more computational power

Grey region: equivalence between two machines is decidable

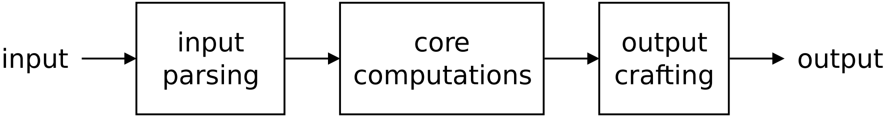
For each class of language there is a machine which required just the computational power needed

Which one would be more useful to an attacker?

Agenda

1. Language theory in a nutshell
2. Model of the network stack... getting weird
3. Principles of secure design
4. Conclusion

Model of a program



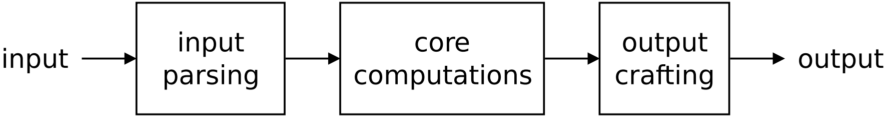
What does a program do?

- take some input
- parse it
- do some computation
- create the output
- send the output

This model includes:

- application inputs
- network stack inputs
- mono-block applications
- multi-blocks applications

Model of a program



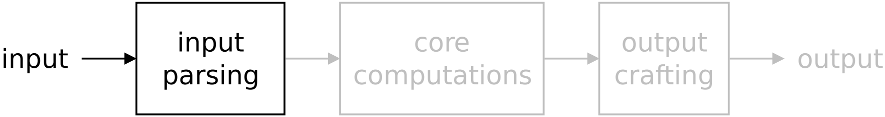
What does a program do?

- take some input
- parse it
- do some computation
- create the output
- send the output

This model includes:

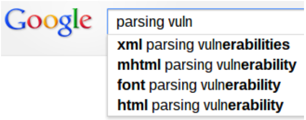
- application inputs
- network stack inputs
- mono-block applications
- multi-blocks applications

Input parsing matters



The paper focus is on input parsing

- does not covers all security problems
- but a lot of them are covered!



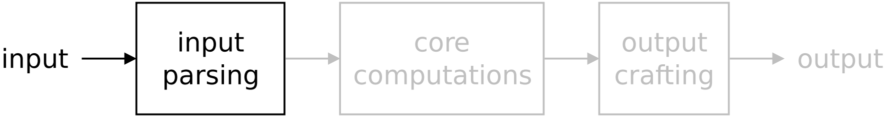
parsing vuln

xml parsing vuln**erabilities**
mhtml parsing vuln**erability**
font parsing vuln**erability**
html parsing vuln**erability**

Remember[2] Chrome locking the rendering engine (including input parsing) inside a sandbox?

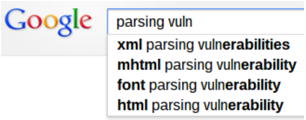
[2] Adam Barth and al., The Security Architecture of the Chromium Browser http://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf

Input parsing matters



The paper focus is on input parsing

- does not covers all security problems
- but a lot of them are covered!


parsing vuln

xml parsing vuln**erabilities**

mhtml parsing vuln**erability**

font parsing vuln**erability**

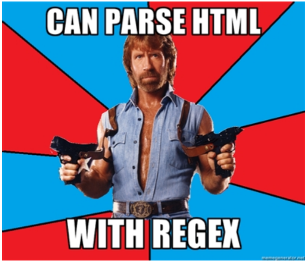
html parsing vuln**erability**

Remember[2] Chrome locking the rendering engine (including input parsing) inside a sandbox?

[2] Adam Barth and al., The Security Architecture of the Chromium Browser http://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf

In the real world

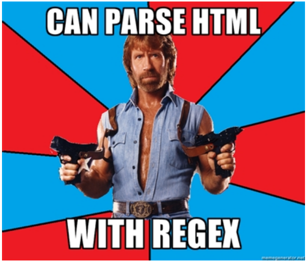
How input parsing is done in the real world?



- spread in the code
- handwritten recognisers
  - faulty
- usually REGEX
  - of course, not just for regular languages
  - easy to get wrong

Are open standards helping?

In the real world

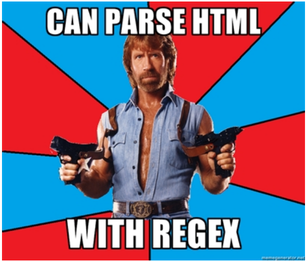


How input parsing is done in the real world?

- spread in the code
- handwritten recognisers
  - faulty
- usually REGEX
  - of course, not just for regular languages
  - easy to get wrong

Are open standards helping?

In the real world

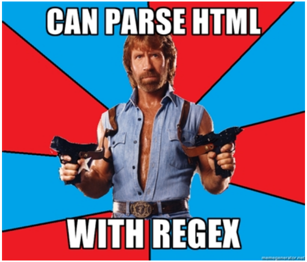


How input parsing is done in the real world?

- spread in the code
- handwritten recognisers
  - faulty
- usually REGEX
  - of course, not just for regular languages
  - easy to get wrong

Are open standards helping?

In the real world

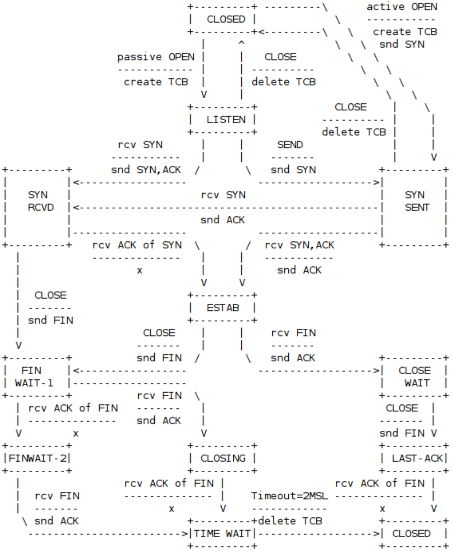


How input parsing is done in the real world?

- spread in the code
- handwritten recognisers
  - faulty
- usually REGEX
  - of course, not just for regular languages
  - easy to get wrong

Are open standards helping?

RFC 793: TCP



```
                              +---------+ ---------\      active OPEN
                              |  CLOSED |            \    -----------
                              +---------+<---------\   \   create TCB
                                |     ^              \   \  snd SYN
                   passive OPEN |     |   CLOSE        \   \
                   ------------ |     | ----------       \   \
                    create TCB  |     | delete TCB         \   \
                                V     |                      \   \
                              +---------+            CLOSE    |    \
                              |  LISTEN |          ---------- |     |
                              +---------+          delete TCB |     |
                   rcv SYN      |     |     SEND              |     |
                  -----------   |     |    -------            |     V
 +---------+      snd SYN,ACK  /       \   snd SYN          +---------+
 |         |<-----------------           ------------------>|         |
 |   SYN   |                    rcv SYN                     |   SYN   |
 |   RCVD  |<-----------------------------------------------|   SENT  |
 |         |                    snd ACK                     |         |
 |         |------------------           -------------------|         |
 +---------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +---------+
   |           --------------   |     |   -----------
   |                  x         |     |     snd ACK
   |                            V     V
   |  CLOSE                   +---------+
   | -------                  |  ESTAB  |
   | snd FIN                  +---------+
   |                   CLOSE    |     |    rcv FIN
   V                  -------   |     |    -------
 +---------+          snd FIN  /       \   snd ACK          +---------+
 |  FIN    |<-----------------           ------------------>|  CLOSE  |
 | WAIT-1  |------------------                              |   WAIT  |
 +---------+          rcv FIN  \                            +---------+
   | rcv ACK of FIN   -------   |                            CLOSE  |
   | --------------   snd ACK   |                           ------- |
   V        x                   V                           snd FIN V
 +---------+                  +---------+                   +---------+
 |FINWAIT-2|                  | CLOSING |                   | LAST-ACK|
 +---------+                  +---------+                   +---------+
   |                rcv ACK of FIN |                 rcv ACK of FIN |
   |  rcv FIN       -------------- |    Timeout=2MSL -------------- |
   |  -------              x       V    ------------        x       V
    \ snd ACK                 +---------+delete TCB         +---------+
     ------------------------>|TIME WAIT|------------------>| CLOSED  |
                              +---------+                   +---------+
```

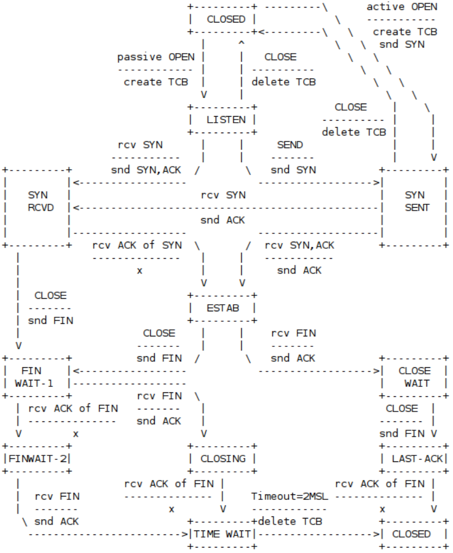
TCP Connection State Diagram

"NOTE BENE: this diagram is only a summary and must not be taken as the total specification."

Where is the problem?

- no BNF
- manually implemented
- each implementation is different

RFC 793: TCP



```
                              +---------+ ---------\      active OPEN
                              |  CLOSED |            \    -----------
                              +---------+<---------\   \   create TCB
                                |     ^              \   \  snd SYN
                   passive OPEN |     |   CLOSE        \   \
                   ------------ |     | ----------       \   \
                    create TCB  |     | delete TCB         \   \
                                V     |                      \   \
                              +---------+            CLOSE    |    \
                              |  LISTEN |          ---------- |     |
                              +---------+          delete TCB |     |
                   rcv SYN      |     |     SEND              |     |
                  -----------   |     |    -------            |     V
 +---------+      snd SYN,ACK  /       \   snd SYN          +---------+
 |         |<-----------------           ------------------>|         |
 |   SYN   |                    rcv SYN                     |   SYN   |
 |   RCVD  |<-----------------------------------------------|   SENT  |
 |         |                    snd ACK                     |         |
 |         |------------------           -------------------|         |
 +---------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +---------+
   |           --------------   |     |   -----------
   |                  x         |     |     snd ACK
   |                            V     V
   |  CLOSE                   +---------+
   | -------                  |  ESTAB  |
   | snd FIN                  +---------+
   |                   CLOSE    |     |    rcv FIN
   V                  -------   |     |    -------
 +---------+          snd FIN  /       \   snd ACK          +---------+
 |  FIN    |<-----------------           ------------------>|  CLOSE  |
 | WAIT-1  |------------------                              |   WAIT  |
 +---------+          rcv FIN  \                            +---------+
   | rcv ACK of FIN   -------   |                            CLOSE  |
   | --------------   snd ACK   |                           ------- |
   V        x                   V                           snd FIN V
 +---------+                  +---------+                   +---------+
 |FINWAIT-2|                  | CLOSING |                   | LAST-ACK|
 +---------+                  +---------+                   +---------+
   |                rcv ACK of FIN |                 rcv ACK of FIN |
   |  rcv FIN       -------------- |    Timeout=2MSL -------------- |
   |  -------              x       V    ------------        x       V
    \ snd ACK                 +---------+delete TCB         +---------+
     ------------------------>|TIME WAIT|------------------>| CLOSED  |
                              +---------+                   +---------+
```

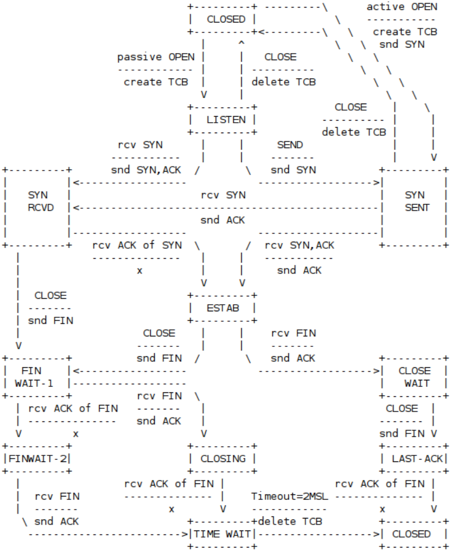
TCP Connection State Diagram

"NOTE BENE: this diagram is only a summary and must not be taken as the total specification."

Where is the problem?

- no BNF
- manually implemented
- each implementation is different

RFC 793: TCP



```
                              +---------+ ---------\      active OPEN
                              |  CLOSED |            \    -----------
                              +---------+<---------\   \   create TCB
                                |     ^              \   \  snd SYN
                   passive OPEN |     |   CLOSE        \   \
                   ------------ |     | ----------       \   \
                    create TCB  |     | delete TCB         \   \
                                V     |                      \   \
                              +---------+            CLOSE    |    \
                              |  LISTEN |          ---------- |     |
                              +---------+          delete TCB |     |
                   rcv SYN      |     |     SEND              |     |
                  -----------   |     |    -------            |     V
 +---------+      snd SYN,ACK  /       \   snd SYN          +---------+
 |         |<-----------------           ------------------>|         |
 |   SYN   |                    rcv SYN                     |   SYN   |
 |   RCVD  |<-----------------------------------------------|   SENT  |
 |         |                    snd ACK                     |         |
 |         |------------------           -------------------|         |
 +---------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +---------+
   |           --------------   |     |   -----------
   |                  x         |     |     snd ACK
   |                            V     V
   |  CLOSE                   +---------+
   | -------                  |  ESTAB  |
   | snd FIN                  +---------+
   |                   CLOSE    |     |    rcv FIN
   V                  -------   |     |    -------
 +---------+          snd FIN  /       \   snd ACK          +---------+
 |  FIN    |<-----------------           ------------------>|  CLOSE  |
 | WAIT-1  |------------------                              |   WAIT  |
 +---------+          rcv FIN  \                            +---------+
   | rcv ACK of FIN   -------   |                            CLOSE  |
   | --------------   snd ACK   |                           ------- |
   V        x                   V                           snd FIN V
 +---------+                  +---------+                   +---------+
 |FINWAIT-2|                  | CLOSING |                   | LAST-ACK|
 +---------+                  +---------+                   +---------+
   |                rcv ACK of FIN |                 rcv ACK of FIN |
   |  rcv FIN       -------------- |    Timeout=2MSL -------------- |
   |  -------              x       V    ------------        x       V
    \ snd ACK                 +---------+delete TCB         +---------+
     ------------------------>|TIME WAIT|------------------>| CLOSED  |
                              +---------+                   +---------+
```

TCP Connection State Diagram

“NOTE BENE: this diagram is only a summary and must not be taken as the total specification.”

Where is the problem?

- no BNF
- manually implemented
- each implementation is different

Are RFC helping?

What if... we don't use them?

$ man 7 tcp

Linux uses the BSD compatible interpretation of the urgent pointer field by default. **This violates RFC 1122**, but is required for interoperability with other stacks. It can be changed via /proc/sys/net/ipv4/tcp_stdurg.

So what do we have?

- Various implementations
- Different behaviours

How can this be used?

Are RFC helping?

What if... we don't use them?

> **$ man 7 tcp**
>
> Linux uses the BSD compatible interpretation of the urgent pointer field by default. **This violates RFC 1122**, but is required for interoperability with other stacks. It can be changed via /proc/sys/net/ipv4/tcp_stdurg.

So what do we have?

- Various implementations
- Different behaviours

How can this be used?

Are RFC helping?

What if... we don't use them?

> **$ man 7 tcp**
>
> Linux uses the BSD compatible interpretation of the urgent pointer field by default. **This violates RFC 1122**, but is required for interoperability with other stacks. It can be changed via /proc/sys/net/ipv4/tcp_stdurg.

So what do we have?

- Various implementations
- Different behaviours

How can this be used?

Exploiting different parsers for the same protocol

Fingerprinting

- detect differences in implementations (dialects)
- xprobe, nmap...

Real exploits

- use differences in implementations
- IDS evasion
- 0day hunting using...

Exploiting different parsers for the same protocol

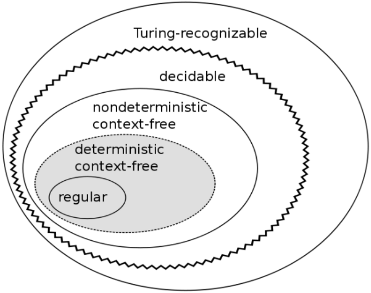
Fingerprinting

- detect differences in implementations (dialects)
- xprobe, nmap...

Real exploits

- use differences in implementations
- IDS evasion
- 0day hunting using...

Parse tree differential analysis



Chomsky hierarchy

Pick different parsers of the same protocol

- compare their parse tree
- if they are different, you probably have a 0 day

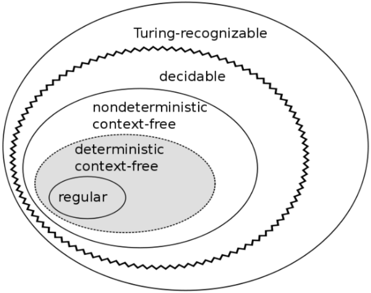
Outside the grey area, automaton equivalence is undecidable

- but we can still find differences!

Result

- they found *clusters* of 0 days
- let's look at an example

Parse tree differential analysis



Chomsky hierarchy

Pick different parsers of the same protocol

- compare their parse tree
- if they are different, you probably have a 0 day

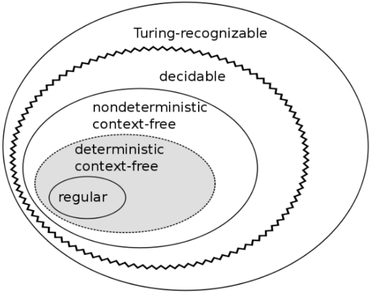
Outside the grey area, automaton equivalence is undecidable

- but we can still find differences!

Result

- they found *clusters* of 0 days
- let's look at an example

Parse tree differential analysis



Chomsky hierarchy

Pick different parsers of the same protocol

- compare their parse tree
- if they are different, you probably have a 0 day

Outside the grey area, automaton equivalence is undecidable

- but we can still find differences!

Result

- they found *clusters* of 0 days
- let's look at an example

Example: exploit on X.509 parsing (2009)

Craft a certificate with

```
CN=www.mywebsite.com/CN=www.bank.com/CN=*
```

OpenSSL parser was only considering `www.mywebsite.com`

- used by CA to sign the certificate

IE parser was only considering `www.bank.com`

- you just got a signed certificate for www.bank.com
- MITM on SSL connections

Firefox was *only* considering $*$ (i.e. all possible names)

- *all your base are belong to us*

Example: exploit on X.509 parsing (2009)

Craft a certificate with

```
CN=www.mywebsite.com/CN=www.bank.com/CN=*
```

OpenSSL parser was only considering `www.mywebsite.com`

- used by CA to sign the certificate

IE parser was only considering `www.bank.com`

- you just got a signed certificate for www.bank.com
- MITM on SSL connections

Firefox was *only* considering $*$ (i.e. all possible names)

- *all your base are belong to us*

Example: exploit on X.509 parsing (2009)

Craft a certificate with

```
CN=www.mywebsite.com/CN=www.bank.com/CN=*
```

OpenSSL parser was only considering `www.mywebsite.com`

- used by CA to sign the certificate

IE parser was only considering `www.bank.com`

- you just got a signed certificate for www.bank.com
- MITM on SSL connections

Firefox was *only* considering $*$ (i.e. all possible names)

- *all your base are belong to us*

Example: exploit on X.509 parsing (2009)

Craft a certificate with

```
CN=www.mywebsite.com/CN=www.bank.com/CN=*
```

OpenSSL parser was only considering `www.mywebsite.com`

- used by CA to sign the certificate

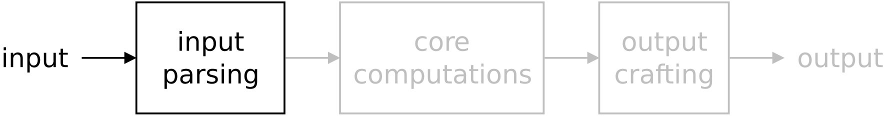
IE parser was only considering `www.bank.com`

- you just got a signed certificate for www.bank.com
- MITM on SSL connections

Firefox was *only* considering $*$ (i.e. all possible names)

- *all your base are belong to us*

Back to our model

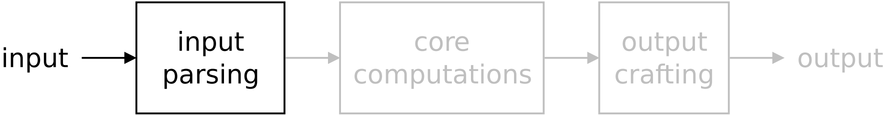


Assume that there is a vulnerability in the input parser.

What will you do as an attacker?

- craft specific inputs that exploit this vulnerability
- take control of the machine running the parser
- perform arbitrary computations

Back to our model

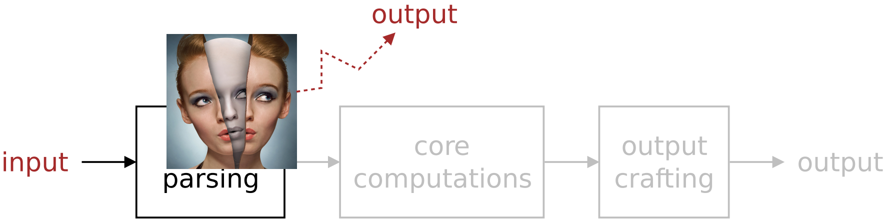


Assume that there is a vulnerability in the input parser.

What will you do as an attacker?

- craft specific inputs that exploit this vulnerability
- take control of the machine running the parser
- perform arbitrary computations

A weird machine rears its head



output

input

parsing

core computations

output crafting

output

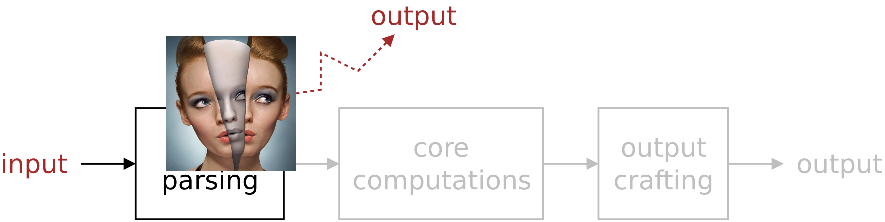
Formalising:

- using an unexpected language having side-effects
- exploiting the *weird machine*

You have already done that. . . Particularly true for the following:

- (blind) SQL injection
- ROP!!!

A weird machine rears its head



output

input

parsing

core computations

output crafting

output

Formalising:

- using an unexpected language having side-effects
- exploiting the *weird machine*

You have already done that. . . Particularly true for the following:

- (blind) SQL injection
- ROP!!!

Mitigating weird machine exploitation

Don't give to the parser more computational power than needed

- the attacker will be less powerful
- FGPA / correctness proof

Obviously, only works if the parser does not need to be Turing-complete...

Do you think it is practical?

Mitigating weird machine exploitation

Don't give to the parser more computational power than needed

- the attacker will be less powerful
- FGPA / correctness proof

Obviously, only works if the parser does not need to be Turing-complete. . .

Do you think it is practical?

Mitigating weird machine exploitation

Don't give to the parser more computational power than needed

- the attacker will be less powerful
- FGPA / correctness proof

Obviously, only works if the parser does not need to be Turing-complete…

Do you think it is practical?

Mitigating weird machine exploitation

Don't give to the parser more computational power than needed

- the attacker will be less powerful
- FGPA / correctness proof

Obviously, only works if the parser does not need to be Turing-complete...

Do you think it is practical?

Agenda

1. Language theory in a nutshell
2. Model of the network stack. . . getting weird
3. Principles of secure design
4. Conclusion

Principles of secure design

Definition (Principle 1)

Request and grant minimal computational power

Definition (Principle 2)

Secure composition requires parser computational equivalence

Ok, what does that mean?

What are you supposed to do:

- when you implement a protocol?
- when you design a protocol?

Principles of secure design

Definition (Principle 1)

Request and grant minimal computational power

Definition (Principle 2)

Secure composition requires parser computational equivalence

Ok, what does that mean?

What are you supposed to do:

- when you implement a protocol?
- when you design a protocol?

Principles of secure design

Definition (Principle 1)

Request and grant minimal computational power

Definition (Principle 2)

Secure composition requires parser computational equivalence

Ok, what does that mean?

What are you supposed to do:

- when you implement a protocol?
- when you design a protocol?

Principles of secure design

Definition (Principle 1)

Request and grant minimal computational power

Definition (Principle 2)

Secure composition requires parser computational equivalence

Ok, what does that mean?

What are you supposed to do:

- when you implement a protocol?
- when you design a protocol?

Principle of secure design for implementation

Definition (Principle 1)

~~Request and~~ grant minimal computational power

If a weird machine appears, at least the damages are limited (if any)

Definition (Principle 2)

Secure composition requires parser computational equivalence

What does it means here? Makes the review easier:

- create the parser automatically from the BNF of the protocol (if any)
- isolate the input parsing from the rest of your program

Principle of secure design for implementation

Definition (Principle 1)

~~Request and~~ grant minimal computational power

If a weird machine appears, at least the damages are limited (if any)

Definition (Principle 2)

Secure composition requires parser computational equivalence

What does it means here? Makes the review easier:

- create the parser automatically from the BNF of the protocol (if any)
- isolate the input parsing from the rest of your program

Principle of secure design for implementation

Definition (Principle 1)

~~Request and~~ grant minimal computational power

If a weird machine appears, at least the damages are limited (if any)

Definition (Principle 2)

Secure composition requires parser computational equivalence

What does it means here? Makes the review easier:

- create the parser automatically from the BNF of the protocol (if any)
- isolate the input parsing from the rest of your program

Principle 1 of secure design for protocol design

Definition (Principle 1)

Request ~~and grant~~ minimal computational power

How much power does your protocol really needs?

- do you really needs those length fields?
  - would make your protocol at least context-sensitive
  - could you use S-expressions instead? (context-free)

Avoid the halting problem of network stack insecurity

- don't create recursively enumerable protocols
- otherwise parsing is undecidable

It's easy to give too much computational power

- PDFs have build-in JS support
- HTML + CSS3 is (very close) to Turing-completeness

Principle 1 of secure design for protocol design

Definition (Principle 1)

Request ~~and grant~~ minimal computational power

How much power does your protocol really needs?

- do you really needs those length fields?
  - would make your protocol at least context-sensitive
  - could you use S-expressions instead? (context-free)

Avoid the halting problem of network stack insecurity

- don't create recursively enumerable protocols
- otherwise parsing is undecidable

It's easy to give too much computational power

- PDFs have build-in JS support
- HTML + CSS3 is (very close) to Turing-completeness

Principle 1 of secure design for protocol design

Definition (Principle 1)

Request ~~and grant~~ minimal computational power

How much power does your protocol really needs?

- do you really needs those length fields?
  - would make your protocol at least context-sensitive
  - could you use S-expressions instead? (context-free)

Avoid the halting problem of network stack insecurity

- don't create recursively enumerable protocols
- otherwise parsing is undecidable

It's easy to give too much computational power

- PDFs have build-in JS support
- HTML + CSS3 is (very close) to Turing-completeness

Principle 1 of secure design for protocol design

Definition (Principle 1)

Request ~~and grant~~ minimal computational power

How much power does your protocol really needs?

- do you really needs those length fields?
  - would make your protocol at least context-sensitive
  - could you use S-expressions instead? (context-free)

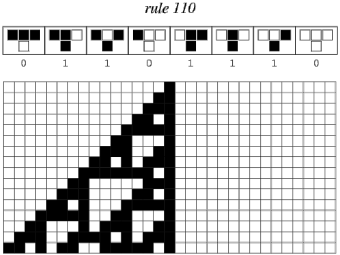
Avoid the halting problem of network stack insecurity

- don't create recursively enumerable protocols
- otherwise parsing is undecidable

It's easy to give too much computational power

- PDFs have build-in JS support
- HTML + CSS3 is (very close) to Turing-completeness

Rule 110



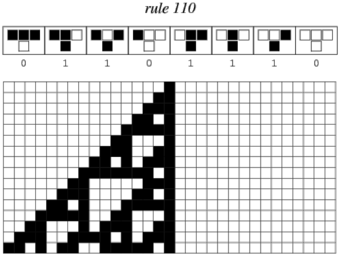
What is Rule 110?

- you have a infinite array filled with 0s and 1s
- at each iteration, change each cell n depending on the values of cells $(n-1,\ n,\ n+1)$ of the previous iteration
- is Turing-complete

Implementation of Rule 110 in HTML + CSS3

- needs some basic interaction from the user
- is obviously not working on infinite arrays
- so HTML + CSS3 is almost Turing-complete
- https://github.com/elitheeli/stupid-machines/blob/master/rule110/rule110-full.html

Rule 110



What is Rule 110?

- you have a infinite array filled with 0s and 1s
- at each iteration, change each cell *n* depending on the values of cells $(n-1,\ n,\ n+1)$ of the previous iteration
- is Turing-complete

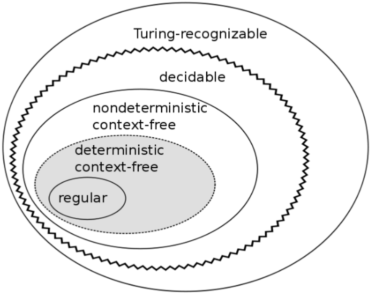
Implementation of Rule 110 in HTML + CSS3

- needs some basic interaction from the user
- is obviously not working on infinite arrays
- so HTML + CSS3 is almost Turing-complete
- https://github.com/elitheeli/stupid-machines/blob/master/rule110/rule110-full.html

Principle 2 of secure design for protocol design

Definition (Principle 2)

Secure composition requires parser computational equivalence



Chomsky hierarchy

Parser equivalence is only possible to check for the grey categories

- see principle 1

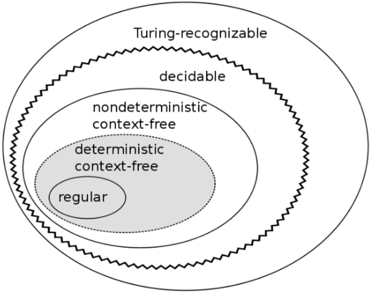
Create BNF of your protocol

- so that implementations can use them
- and checking computational equivalence is easier

Principle 2 of secure design for protocol design

Definition (Principle 2)

Secure composition requires parser computational equivalence



Chomsky hierarchy

Parser equivalence is only possible to check for the grey categories

- see principle 1

Create BNF of your protocol

- so that implementations can use them
- and checking computational equivalence is easier

About Postel's law

Definition (Postel's law (or robustness principle), from RFC 793)

Be conservative in what you do, be liberal in what you accept from others.

- Trade security for laziness
- Goes against principle 2 (no equivalence between parsers)
- It's even easier to use a very strict input parser, if we have the corresponding BNF

Was from 1981... maybe it's time to patch it?

About Postel's law

Definition (Postel's law (or robustness principle), from RFC 793)

Be conservative in what you do, be liberal in what you accept from others.

- Trade security for laziness
- Goes against principle 2 (no equivalence between parsers)
- It's even easier to use a very strict input parser, if we have the corresponding BNF

Was from 1981... maybe it's time to patch it?

About Postel's law

Definition (Postel's law (or robustness principle), from RFC 793)

Be conservative in what you do, be liberal in what you accept from others.

- Trade security for laziness
- Goes against principle 2 (no equivalence between parsers)
- It's even easier to use a very strict input parser, if we have the corresponding BNF

Was from 1981... maybe it's time to patch it?

About Postel's law

Definition (Postel's law (or robustness principle), from RFC 793)

Be conservative in what you do, be liberal in what you accept from others.

- Trade security for laziness
- Goes against principle 2 (no equivalence between parsers)
- It's even easier to use a very strict input parser, if we have the corresponding BNF

Was from 1981... maybe it's time to patch it?

About Postel's law

Definition (Postel's law (or robustness principle), from RFC 793)

Be conservative in what you do, be liberal in what you accept from others.

- Trade security for laziness
- Goes against principle 2 (no equivalence between parsers)
- It's even easier to use a very strict input parser, if we have the corresponding BNF

Was from 1981... maybe it's time to patch it?

The Postel's law patch

```
--- ietf/postels-principle
+++ ietf/postels-principle
- Be liberal about what you accept.
+ Be definite about what you accept.(*)
+
+ Treat inputs as a language, accept it with a matching computational
+ power, generate its recognizer from its grammar.
+
+ Treat input-handling computational power as privilege, and reduce it
+ whenever possible.
+
+
+ (*) For the sake of your users, be definite about what you accept.
+ Being liberal worked best for simpler protocols and languages,
+ and is in fact limited to such languages; be sure to keep your
+ language regular or at most context free (no length fields).
+ Being more liberal did not work so well for early IPv4 stacks:
+ they were initially vulnerable to weak packet parser attacks, and
+ ended up eliminating many options and features from normal use.
+ Furthermore, presence of these options in traffic came to be regarded
+ as a sign of suspicious or malicious activities, to be mitigated by
+ traffic normalization or outright rejection. At current protocol
+ complexities, being liberal actually means exposing the users of your
+ software to intractable or malicious computations.
```

http://www.cs.dartmouth.edu/∼sergey/langsec/postel-principle-patch.txt

Agenda

1 Language theory in a nutshell

2 Model of the network stack… getting weird

3 Principles of secure design

4 Conclusion

What have we done so far

Formalise network stack

- (partial) explanation of fingerprinting
- method to find 0 days (parse tree differential analysis)

Principles of secure design

- for protocol designs
- for parser implementations
- let's just walk through them again

What have we done so far

Formalise network stack

- (partial) explanation of fingerprinting
- method to find 0 days (parse tree differential analysis)

Principles of secure design

- for protocol designs
- for parser implementations
- let's just walk through them again

What have we done so far

Formalise network stack

- (partial) explanation of fingerprinting
- method to find 0 days (parse tree differential analysis)

Principles of secure design

- for protocol designs
- for parser implementations

- let's just walk through them again

Principle 0: full recognition before processing

Principle 1: minimal computational power

Principle 2: parser computational equivalence

List of images used

By order of apparition:

- CC logo by Creative Commons under the public domain http://en.wikipedia.org/wiki/File:Cc.logo.circle.svg
- CC By logo by Sting under CC By 2.5 http://en.wikipedia.org/wiki/File:Cc-by_new.svg
- CC Sa logo by Creative Commons under CC By http://en.wikipedia.org/wiki/File:Cc-sa.svg
- Picture of Emmanuelle Laborit by Penelope1967 under copyright http://www.youtube.com/watch?v=HhCXZ2IX6uQ
- Piet program by Thomas Schoch under GFDL and CC By-Sa-3.0 http://commons.wikimedia.org/wiki/File:Piet_Program.gif
- The Tower of Babel by Pieter Bruegel the Elder under the public domain http://en.wikipedia.org/wiki/File:Pieter_Bruegel_the_Elder_-_The_Tower_of_Babel_(Vienna)_-_Google_Art_Project_-_edited.jpg
- Chomsky hierarchy from the original paper (see first page)
- Model of Turing machine by Rocky Acosta under CC By http://commons.wikimedia.org/wiki/File:Turing_Machine_Model_Davey_2012.jpg
- Abstruse Goose comic about the halting problem under CC By Nc http://abstrusegoose.com/440
- Chuck Noris meme by Memegenerator http://memegenerator.net/instance/34260152
- Who I really am by Tucia under CC By http://www.flickr.com/photos/tucia/4649549850/
- Rule 110 image fby Eric Weisstein from MathWorld –A Wolfram Web Resource http://mathworld.wolfram.com/Rule110.html
- Occupy LangSec images by Kythera of Anevern http://www.cs.dartmouth.edu/∼sergey/langsec/occupy/

For science. . . you weird machine

Want more? langsec.org

Q&A time!