Mind your Language(s)!
A discussion about languages and security

Olivier Levillain & Pierre Chifflier
ANSSI

Hackito Ergo Sum, 2015-10-29

Who are we?

Olivier Levillain (`@pictyeye`)

- 2007-2014 (DCSSI/ANSSI) in the labs (systems then network)
- since 2015 (ANSSI) head of the training center
- PhD student (since 2011!) working on SSL/TLS
- Participation to the languages studies since 2007

Pierre Chifflier (`@pollux7`)

- 2011-2015 (ANSSI) in the labs (systems)
- since 2015 (ANSSI) head of the research Lab for Exploration and Detection (LED)
- Firewalls, IDS, UEFI, compilers, languages, ...

ANSSI

ANSSI (French Network and Information Security Agency) has InfoSec (and no Intelligence) missions:

- detect and early react to cyber attacks
- prevent threats by supporting the development of trusted products and services
- provide reliable advice and support
- communicate on information security threats and the related means of protection

These missions concern:

- governmental entities
- companies
- the general public

Why would we mind our languages?

In 2005, the DCSSI was asked whether JAVA could be used to develop security products or not

The question is interesting, and it can be broadened:

- Are some languages better suited for security? On which criteria?
- Should we forbid, discourage, recommend or require the use of particular languages or particular constructions?
- What would be a language dedicated to security like? What about its compiler and its runtime?

It seems few people considered this question

Foreword

What this presentation is about

- the impact of the language on security properties is understudied
- it covers a broad spectrum of subjects

- since 2005, two studies: JavaSec and LaFoSec (available on `www.ssi.gouv.fr`)
- each time, our partners did not at first share (or even understand) our concerns

- the following examples do not aim at criticising particular languages
- no language was harmed during our work[1]

[1] They were already like that when we began.

The five stages of this presentation

During and after this presentation, you might experience different reactions

- denial: you can check yourself easily most of our examples
- anger: "Of course, language X first converts strings to ints before comparing them. You moron..."
- bargaining: you might be trying to justify the unjustifiable
- depression: "why bother developing if all is lost?"
- acceptance: some languages/constructions are not your friends... you must learn to know them and their quirks

Outline

Outline

[JavaScript] Some are more equal than others

JavaScript offers all the modern comfort...

```
if (0=='0') print("Equal"); else print("Different");

switch (0)
{ case '0':print("Equal");
  default:print("Different");
}
```

[JavaScript] Some are more equal than others

JavaScript offers all the modern comfort...

```
if (0=='0') print("Equal"); else print("Different");

switch (0)
{ case '0':print("Equal");
  default:print("Different");
}
```

Output is `Equal`, then `Different`

[JavaScript] Reconversion

Should we prefer cast and overloading, or associativity and transitivity?

[JavaScript] Reconversion

Should we prefer cast and overloading, or associativity and transitivity?

In JavaScript, `'0'==0` is true, as well as `0=='0.0'`. However, `'0'=='0.0'` is false; in other words, equality is not transitive

[JavaScript] Reconversion

Should we prefer cast and overloading, or associativity and transitivity?

In JavaScript, `'0'==0` is true, as well as `0=='0.0'`. However, `'0'=='0.0'` is false; in other words, equality is not transitive

Another example: the `+` operator, which can be either the addition of integers, or the concatenation of strings, but is associative in both cases

```
a=1; b=2; c='Foo';
print(a+b+c); print(c+a+b); print(c+(a+b));
```

[JavaScript] Reconversion

Should we prefer cast and overloading, or associativity and transitivity?

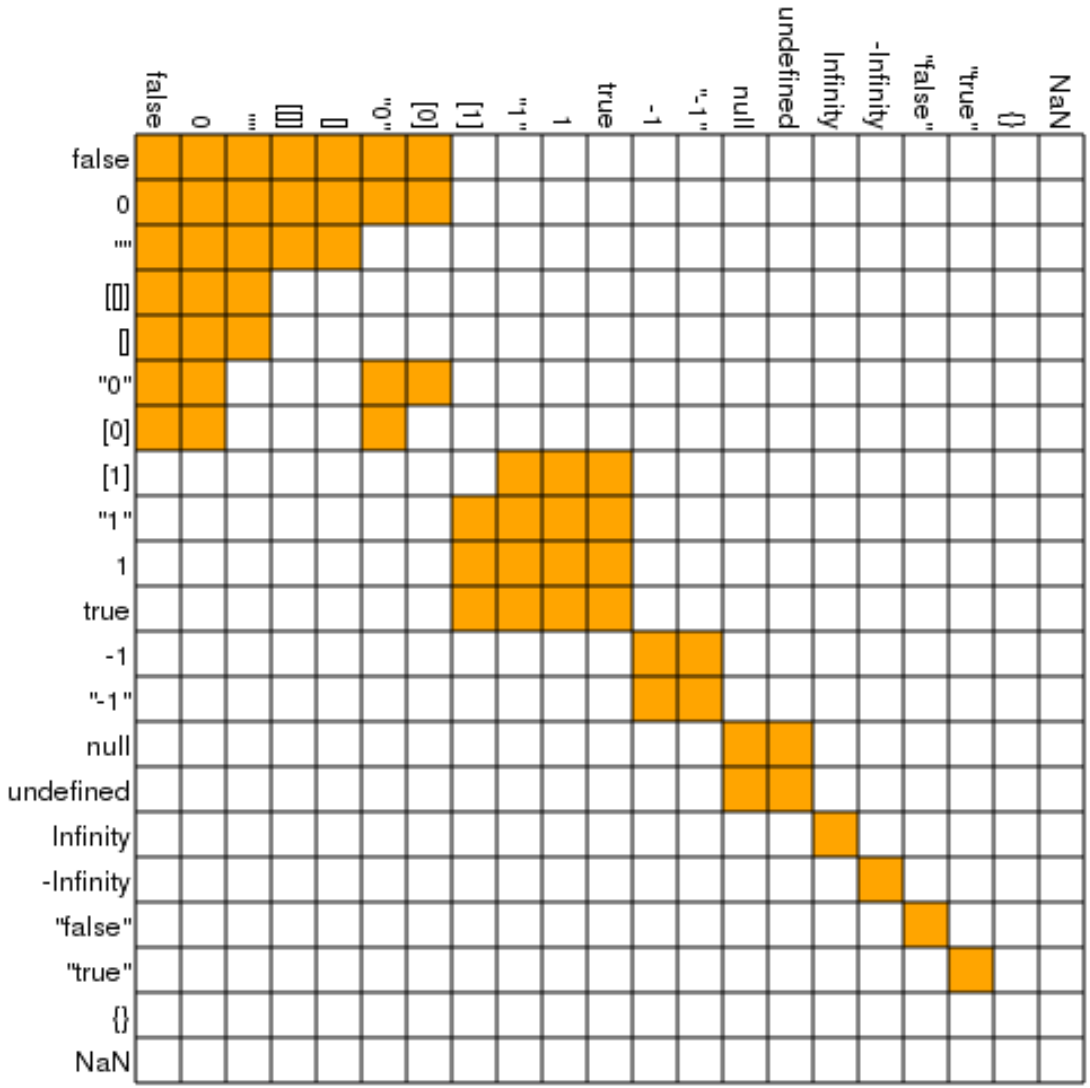
In JavaScript, `'0'==0` is true, as well as `0=='0.0'`. However, `'0'=='0.0'` is false; in other words, equality is not transitive

Another example: the `+` operator, which can be either the addition of integers, or the concatenation of strings, but is associative in both cases

```
a=1; b=2; c='Foo';
print(a+b+c); print(c+a+b); print(c+(a+b));
```

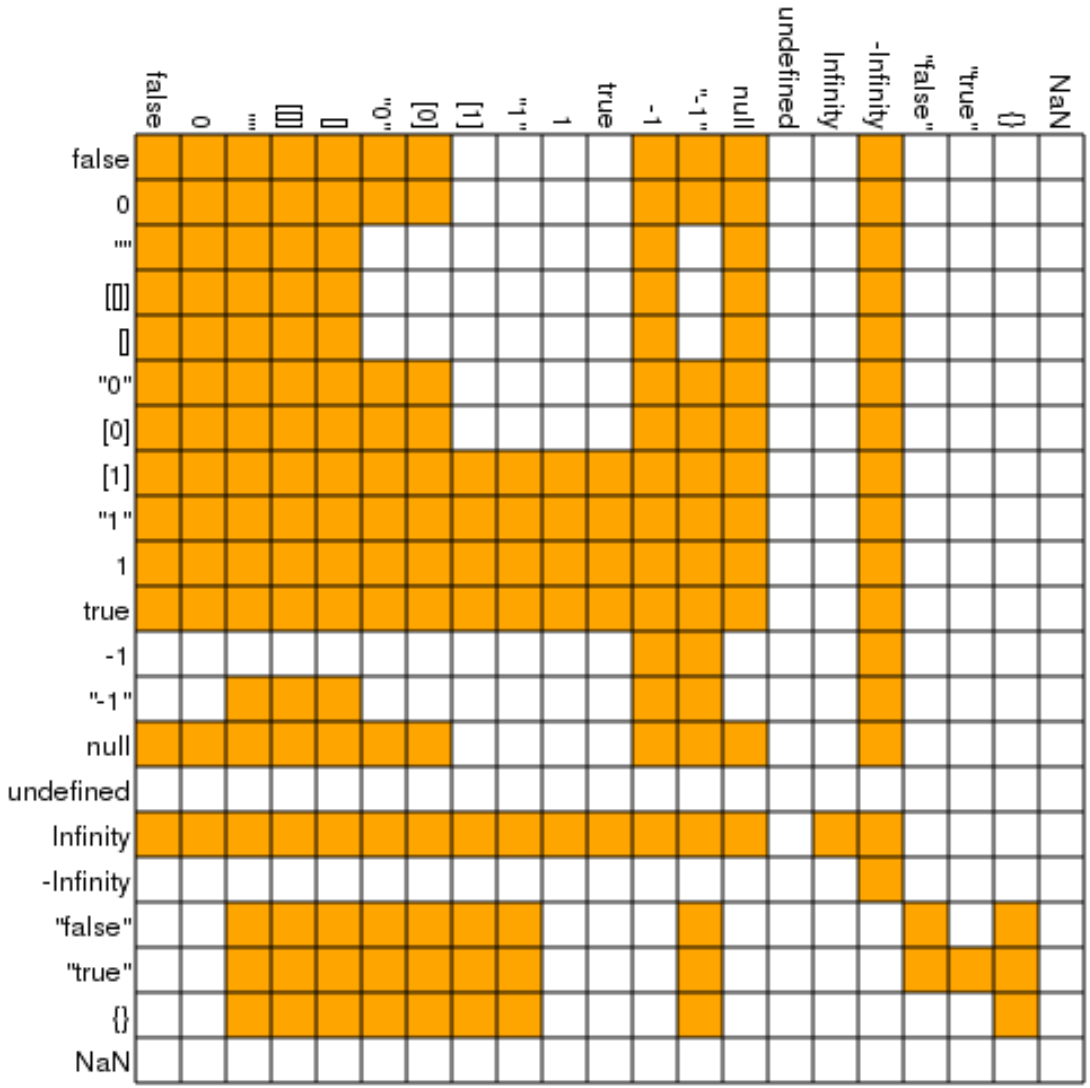
`3Foo`, `Foo12` and `Foo3`

[JavaScript] *Enter the Matrix* 1/4



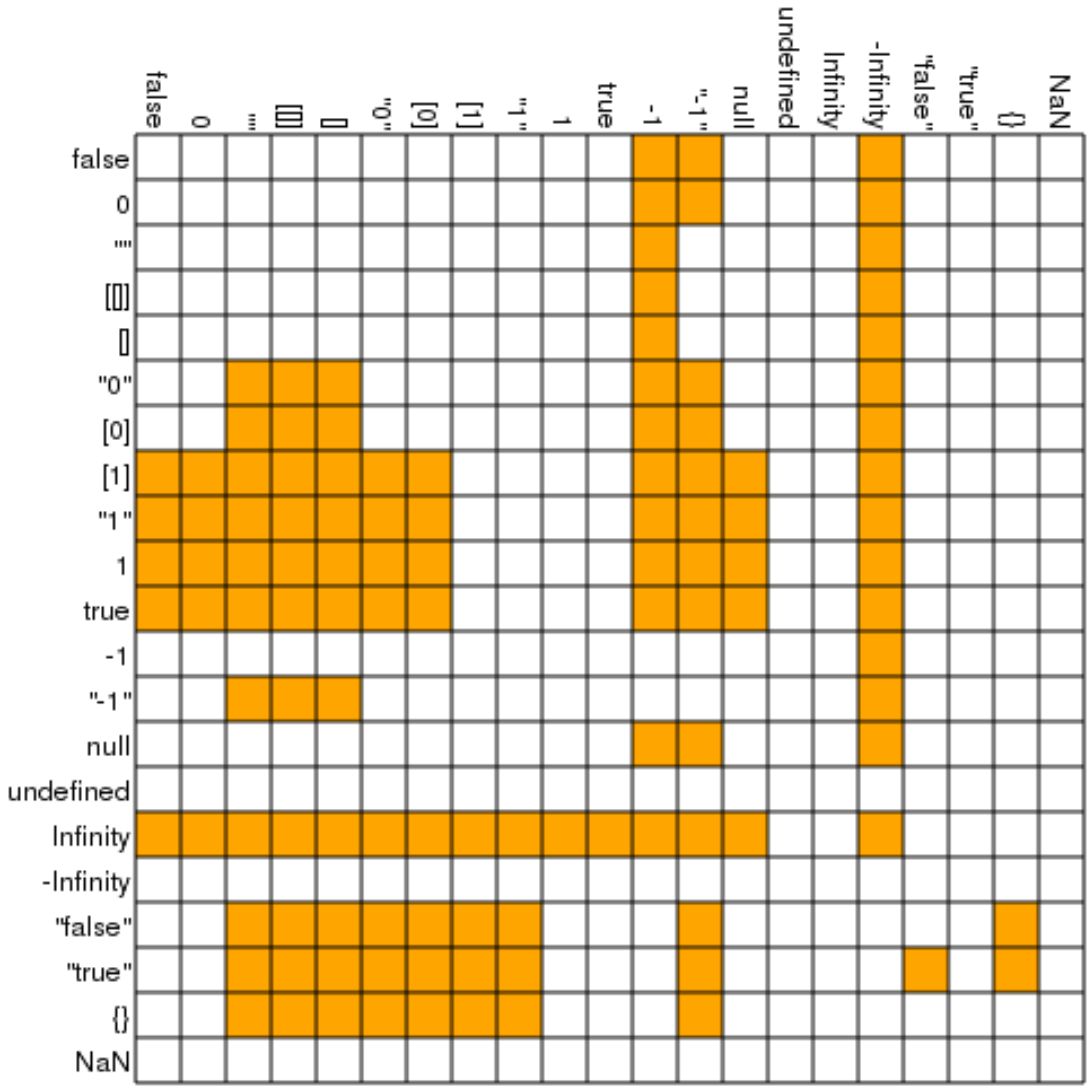
Equal ==

[JavaScript] *Enter the Matrix* 2/4



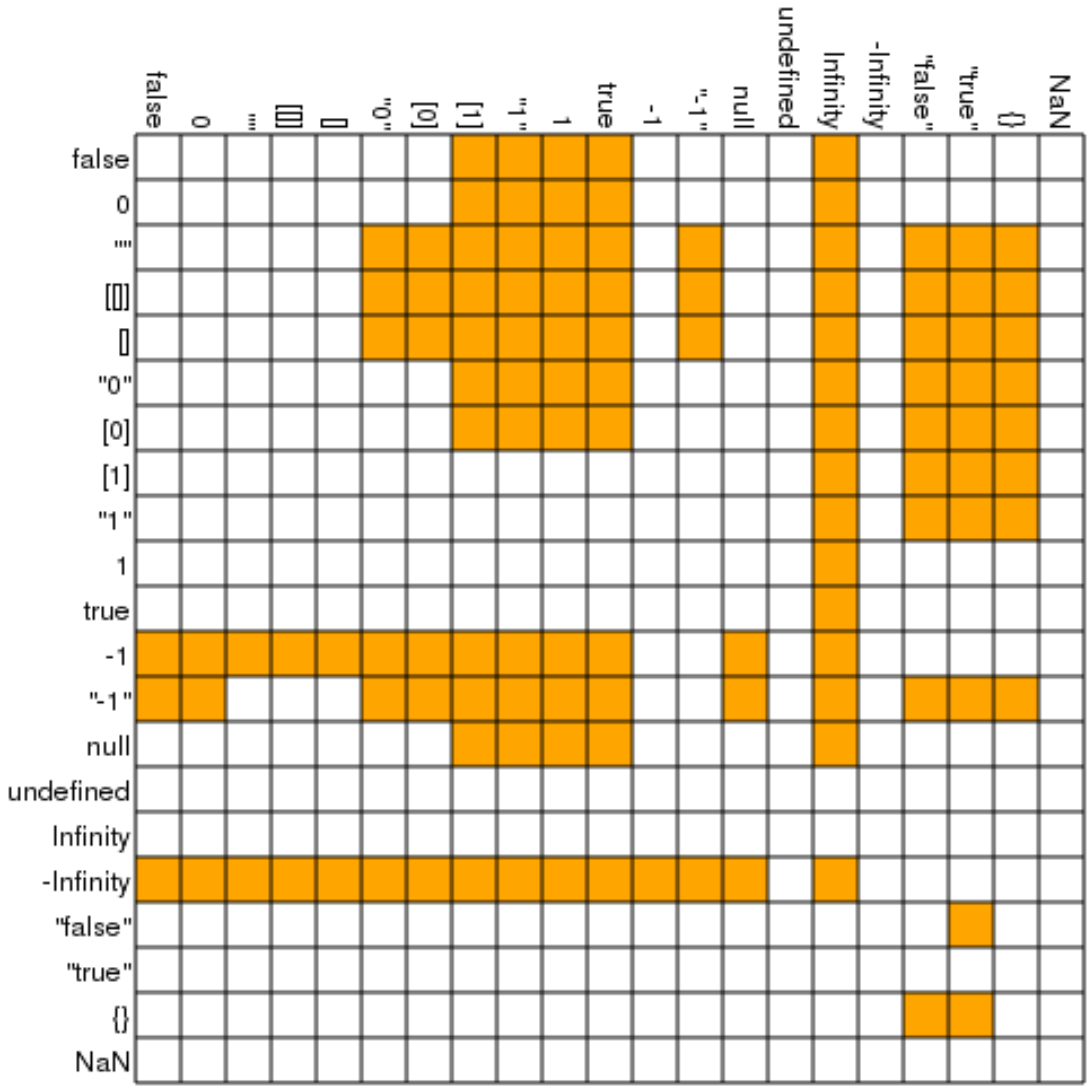
Lesser than or equal <=

[JavaScript] *Enter the Matrix* 3/4

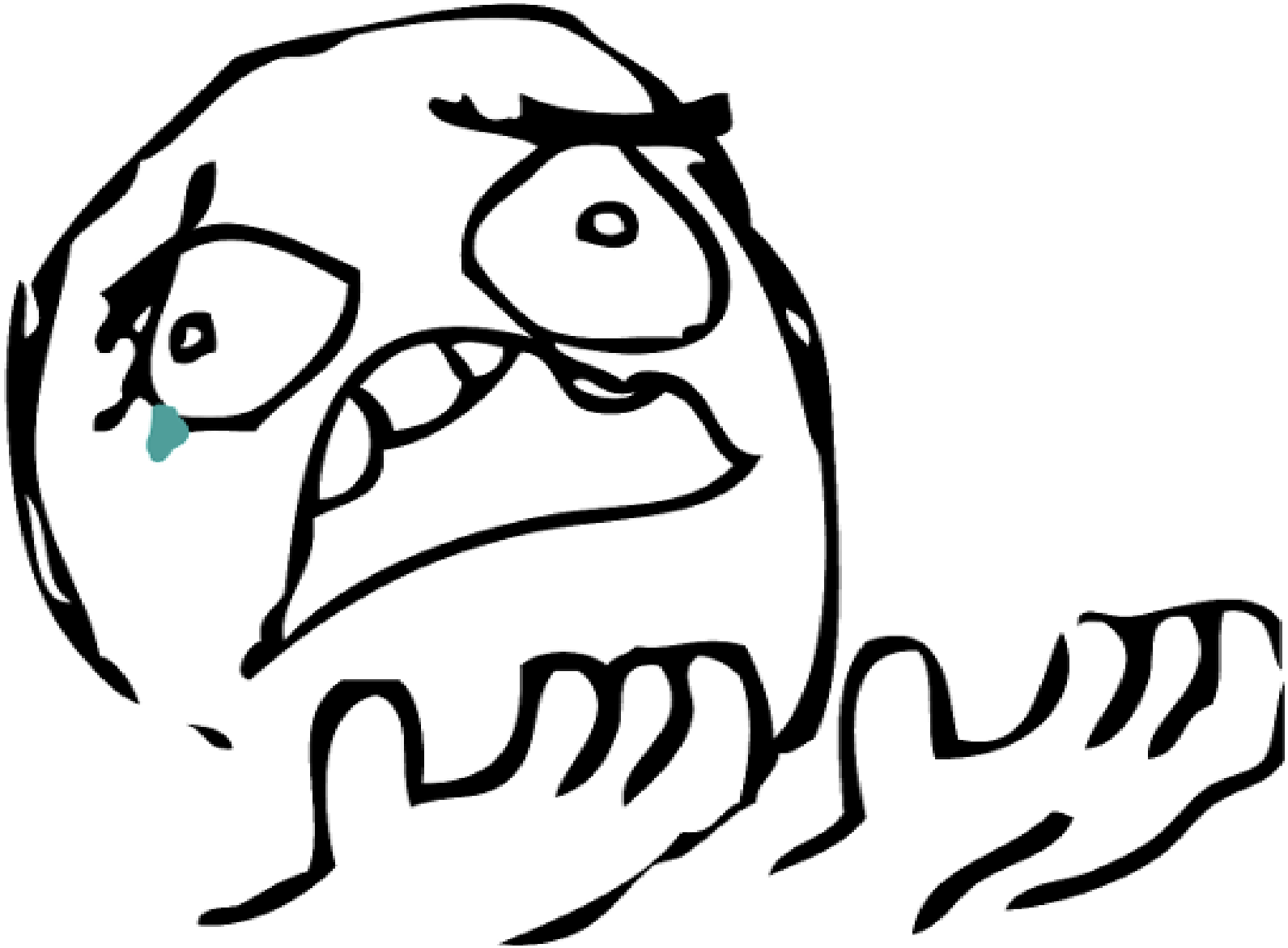


Lesser than <

[JavaScript] *Enter the Matrix* 4/4



Greater than >



[JavaScript] M'enfin

Given that, crypto using JS (in the browser) really looks like a good idea:

- OpenPGP.js
- Google End-To-End
- keybase.io
- Heartbleed and javascript crypto

Outline

[*Shell*] True, False, FILE_NOT_FOUND 1/2

```
#!/bin/bash
PIN=1234
echo -n "Please type your PIN code (4 digits): "
read -s PIN_TYPED; echo

if [ "$PIN" -ne "$PIN_TYPED" ]; then
    echo "Invalid PIN code."; exit 1
else
    echo "Authentication OK"; exit 0
fi
```

[*Shell*] True, False, FILE_NOT_FOUND 1/2

```
#!/bin/bash
PIN=1234
echo -n "Please type your PIN code (4 digits): "
read -s PIN_TYPED; echo

if [ "$PIN" -ne "$PIN_TYPED" ]; then
    echo "Invalid PIN code."; exit 1
else
    echo "Authentication OK"; exit 0
fi
```

A wrong PIN code will be rejected; yet if the user sends non-numeric characters, access will be granted

[C] True, False, FILE_NOT_FOUND 2/2

Focus on the *Goto Fail* vulnerability of GnuTLS (CVE-2014-0092), in March 2014 (lwn.net)

> *But this bug is arguably much worse than Apple's, as it has allowed crafted certificates to evade validation check for all versions of GnuTLS ever released since that project got started in late 2000.[...]*
> *The* `check_if_ca` *function is supposed to return true (any non-zero value in C) or false (zero) depending on whether the issuer of the certificate is a certificate authority (CA). A true return should mean that the certificate passed muster and can be used further, but the bug meant that error returns were misinterpreted as certificate validations.*

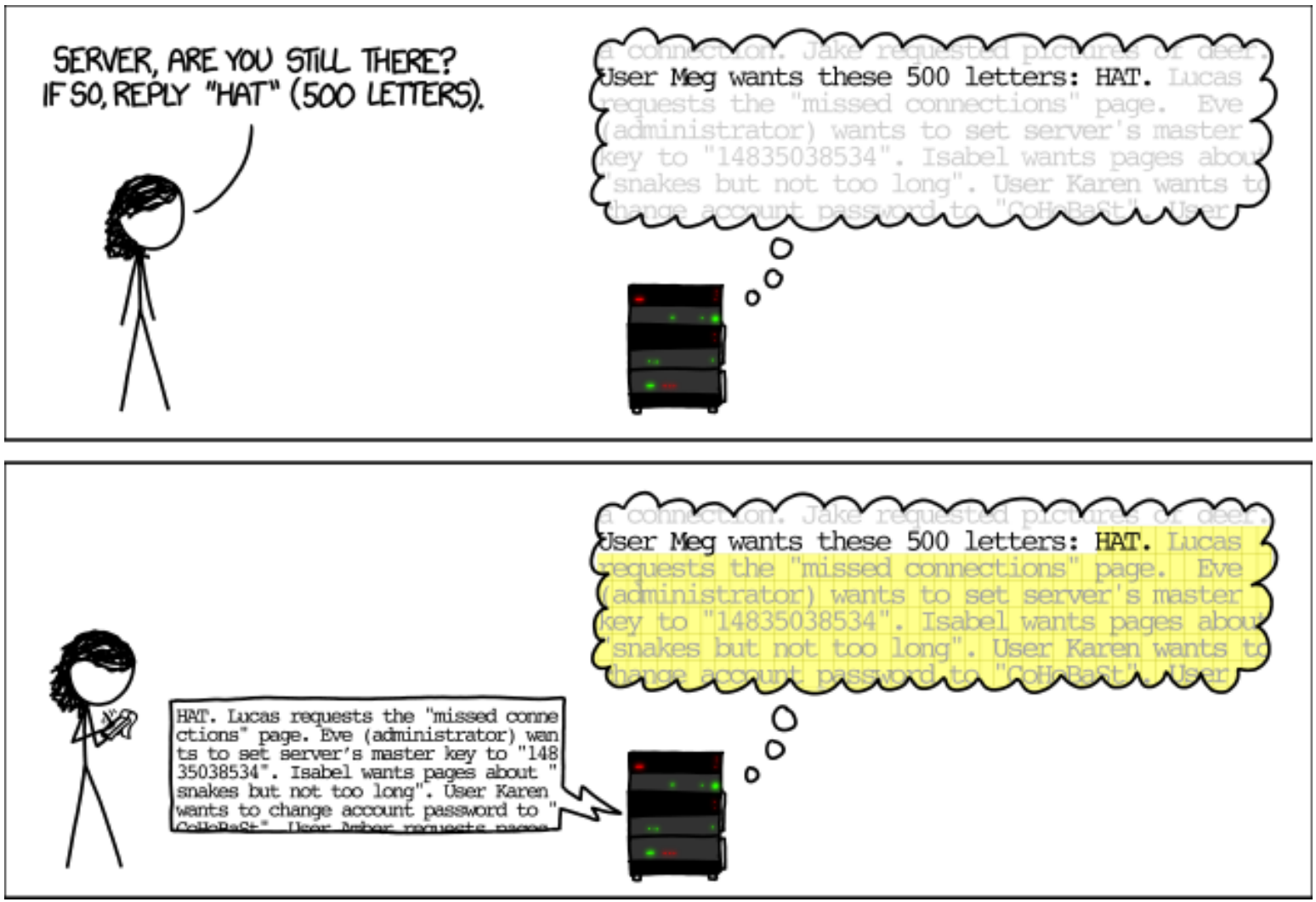
[C] True, False, FILE_NOT_FOUND 2/2

Focus on the *Goto Fail* vulnerability of GnuTLS (CVE-2014-0092), in March 2014 (lwn.net)

> *But this bug is arguably much worse than Apple's, as it has allowed crafted certificates to evade validation check for all versions of GnuTLS ever released since that project got started in late 2000.[...]*
> *The* `check_if_ca` *function is supposed to return true (any non-zero value in C) or false (zero) depending on whether the issuer of the certificate is a certificate authority (CA). A true return should mean that the certificate passed muster and can be used further, but the bug meant that error returns were misinterpreted as certificate validations.*

By the way, a similar bug was found in OpenSSL... in 2008 (CVE-2008-5077).

[C] Echo-logy 1/2



[C] Echo-logy 2/2

The *Heartbleed* vulnerability (CVE-2014-160) was disclosed in April 2014

Concretely, about half of HTTPS servers of the world were impacted, with potential remote compromission of

- private keys
- passwords
- any other information present in the memory of the process...

Using a cryptographic framework has added a vulnerability that was not present, and consequences go way beyond the perimeter of the framework.

The cause was a simple missing test for checking bounds, in the code of a non-critical function of the SSL/TLS protocol.

[C] *Epic Apple's Goto Fail*

Yet another bug in a cryptographic library, revealed in 2014

```
/* Extract from Apple's sslKeyExchange.c */
if ((err=SSLHashSHA1.update(&hashCtx,&serverRandom))!=0)
    goto fail;
if ((err=SSLHashSHA1.update(&hashCtx,&signedParams))!=0)
    goto fail;
    goto fail;
if ((err=SSLHashSHA1.final(&hashCtx,&hashOut))!=0)
    goto fail;
```

Syntax doesn't help, but the compiler doesn't seem concerned about signaling obviously dead code...

[C] Unconditional compromission

A (proposed) Linux kernel modification[2]

```
+ if ((options==(__WCLONE|__WALL)) && (current->uid=0))
+ retval = -EINVAL;
```

[2] Cf. lwn.net/Articles/57135/

[3] To a C programmer strong typing means pressing the keys harder.

[C] Unconditional compromission

A (proposed) Linux kernel modification[2]

```
+ if ((options==(__WCLONE|__WALL)) && (current->uid=0))
+ retval = -EINVAL;
```

Obvious trap : when the test of `options` is true, `current->uid` becomes `0` (*i.e.* the process gains `root` privileges)

The attack is based on the confusion between `=` and `==`, but also on the fact that the affectation returns a value, that C is weakly typed [3] so the integer will be evaluated as a boolean value, that evaluation is lazy, *etc.*

[2] Cf. lwn.net/Articles/57135/

[3] To a C programmer strong typing means pressing the keys harder.

[Java] Varying equality

At least, with physical equality, we know what to expect. . . except in case of subtle interactions with innovating standard libraries

```
Integer a1=42;
Integer a2=42;
if (a1==a2) System.out.println("a1 == a2");

Integer b1=1000;
Integer b2=1000;
if (b1==b2) System.out.println("b1 == b2");
```

[Java] Varying equality

At least, with physical equality, we know what to expect. . . except in case of subtle interactions with innovating standard libraries

```
Integer a1=42;
Integer a2=42;
if (a1==a2) System.out.println("a1 == a2");

Integer b1=1000;
Integer b2=1000;
if (b1==b2) System.out.println("b1 == b2");
```

The output is `a1==a2` (nothing for the second test).
Who wants to guess why?

[Java] Utf? Wtf!

Some compilers are Utf-8-compatible

```
public class Preprocess {
  public static void ma\u0069n (String[] args) {
    if (false==true)
    { //\u000a\u007d\u007b
      System.out.println("Bad things happen!");
    }
  }
}
```

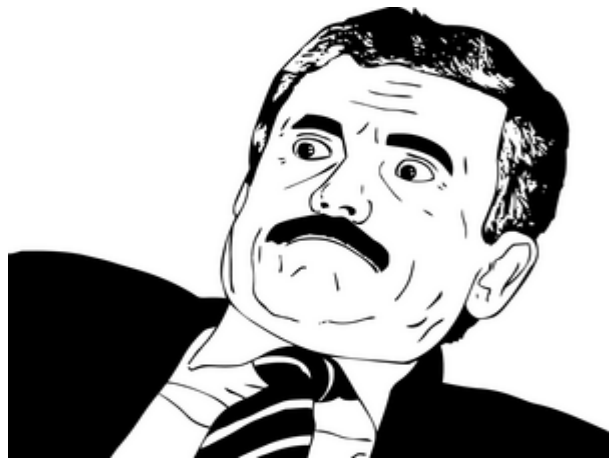
[Java] Utf? Wtf!

Some compilers are Utf-8-compatible

```
public class Preprocess {
  public static void ma\u0069n (String[] args) {
    if (false==true)
    { //\u000a\u007d\u007b
      System.out.println("Bad things happen!");
    }
  }
}
```

The output is obviously `Bad thing happens`: the source code seems to be pre-processed *before* the compilation

L'instant PHP



[PHP] Icônocast

```
$x="2d8"; print($x+1); print("\n");

$x="2d8"; print(++$x."\n"); print(++$x."\n"); print(++$x."\n");
```

[Php] Icônocast

```
$x="2d8"; print($x+1); print("\n");

$x="2d8"; print(++$x."\n"); print(++$x."\n"); print(++$x."\n");
```

The output of the first line is `3` (integer)

[Php] Icônocast

```
$x="2d8"; print($x+1); print("\n");

$x="2d8"; print(++$x."\n"); print(++$x."\n"); print(++$x."\n");
```

The output of the first line is `3` (integer)

Output of the second line is `2d9` (string), `2e0` (string) then `3` (float)

Outline

[Perl] The Perl Jam (31c3) 1/3

Let us consider a simple SQL request in a web application:

```
req = 'select * from users where username=' .
    $dbh->quote ($cgi->param('user'));
```

http://index.cgi?user=user'

With such a request, the quote is properly escaped, as expected, since `$cgi->param('user')` is the string `"user'"`

[Perl] The Perl Jam (31c3) 1/3

Let us consider a simple SQL request in a web application:

```
req = 'select * from users where username=' .
    $dbh->quote ($cgi->param('user'));
```

http://index.cgi?user=user'

With such a request, the quote is properly escaped, as expected, since `$cgi->param('user')` is the string `"user'"`

http://index.cgi?user='or''='&user=3 ?

What about this one, where `$cgi->param('user')` becomes an array with two values?

[Perl] The Perl Jam (31c3) 2/3

Let's look at the quote source code:

```
sub quote ($$; $) {
    my ($self, $str, type) = @_;
    ...
    defined $type && ($type == DBI::SQL_NUMERIC() ... )
       and return $str;
    ... }
```

- When the user parameter is repeated, the framework produces an array instead of a litteral string

[Perl] The Perl Jam (31c3) 2/3

Let's look at the `quote` source code:

```
sub quote ($$; $) {
    my ($self, $str, type) = @_;
    ...
    defined $type && ($type == DBI::SQL_NUMERIC() ... )
       and return $str;
    ... }
```

- When the `user` parameter is repeated, the framework produces an array instead of a litteral string
- In the function, the array is seen as two arguments, instead of one!

[Perl] The Perl Jam (31c3) 2/3

Let's look at the quote source code:

```
sub quote ($$; $) {
    my ($self, $str, type) = @_;
    ...
    defined $type && ($type == DBI::SQL_NUMERIC() ... )
       and return $str;
    ... }
```

- When the user parameter is repeated, the framework produces an array instead of a litteral string
- In the function, the array is seen as two arguments, instead of one!
- Cherry on the cake, SQL_NUMERIC (3) as a second arg allows to bypass the security mechanism

[Perl] The Perl Jam (31c3) 3/3

Type confusion *within the language* is bad, and lead to real security issues.

A solution would be to check the type of the provided argument

But should we *really* expect developers to jump through hoops, *simply* to access function arguments?

[Python] Locality fun

Python offers syntactic constructions equivalent to the classical `map` algorithm on lists, and list comprehensions

```
>>> l = [s+1 for s in [1,2,3]]
>>> l
[2, 3, 4]
```

What happens then if we type `s` into the prompt ?

[Python] Locality fun

Python offers syntactic constructions equivalent to the classical `map` algorithm on lists, and list comprehensions

```
>>> l = [s+1 for s in [1,2,3]]
>>> l
[2, 3, 4]
```

What happens then if we type `s` into the prompt ?

Unless using the latest Python 3 interpreter, `s` is `3`, while the variable `s` should have been local (bound), as in the following snippet:

```
>>> l = map (lambda s : s+1, [1,2,3])
```

[Python] A false midnight (lwn.net/Articles/590299/)

Python allows to use almost anything as a condition in an `if` statement

```
def check_time (start_time, end_time):
    time = datetime.now().time()
    if start_time and end_time:
        return (start_time <= time) and (time <= end_time)
    else:
        return True  # no bounds were specified
```

What should return `check_time (time(23,0,0), time (0, 0, 0))`?

[Python] A false midnight (lwn.net/Articles/590299/)

Python allows to use almost anything as a condition in an `if` statement

```
def check_time (start_time, end_time):
    time = datetime.now().time()
    if start_time and end_time:
        return (start_time <= time) and (time <= end_time)
    else:
        return True  # no bounds were specified
```

What should return `check_time (time(23,0,0), time (0, 0, 0))`?

Since midnight is considered to be false, it's always `True`

[Python] *tuple*-ware

```
>>> foo = ([],)
>>> foo[0] += [1]
TypeError: 'tuple' object does not support item assignment

>>> foo
<<< ([1],)
```

Checking for exceptions **before** doing the action may be an interesting behavior

[Ruby/*Shell*] This is not a *pipe*

In Ruby, `Kernel.open` and `File.open` both allow to open a file, and almost have the same behavior... The first (which is called by `open`) also allows to get the output of a *Shell* command as a file

```
> open ("|ls").each { |x| p x }
"beginend.rb\n"
"beginend.rb~\n"
...
```

On which criteria? The fact that the file name starts with the `|` character

L'instant PHP

[PHP] Get a fix

Extract from `jQuery File Upload Plugin PHP Class`

```
// Fix for overflowing signed 32 bit integers,
// works for sizes up to 2^32-1 bytes (4 GiB - 1):
protected function fix_integer_overflow($size) {
  if ($size < 0) {
    $size += 2.0 * (PHP_INT_MAX + 1);
  }
  return $size;
}

return $this->fix_integer_overflow(filesize($file_path));
```

Outline

[OCaml] *Mutatis mutandis* 1/3

In OCaml, code is static and strings are mutable; but what about strings defined in the code ?

```
let check c =
  if c then "OK" else "KO";;

let f=check false in
  f.[0]<-'O'; f.[1]<-'K';;

check true;;
check false;;
```

[OCaml] *Mutatis mutandis* 1/3

In OCaml, code is static and strings are mutable; but what about strings defined in the code ?

```
let check c =
  if c then "OK" else "KO";;

let f=check false in
  f.[0]<-'O'; f.[1]<-'K';;

check true;;
check false;;
```

Both applications of `check` return `"OK"`

[OCaml] *Mutatis mutandis* 2/3

Previous example is not a redefinition of the `alert` function, but a simple simple side effect; to be convinced, here is the result of this, applied to a function of the standard library

```
let t=string_of_bool false in
  t.[0]<-'t'; t.[1]<-'r'; t.[2]<-'u'; t.[3]<-'e'; t.[4]<-' ';

Printf.printf "1<>1 is %b\n" (1<>1);;
```

Output is `1<>1 is true`

[OCaml] *Mutatis mutandis* 3/3

Other interesting functions are impacted by such string manipulations

Exceptions

- Many standard libraries throw `Failure` exceptions containing a *constant* string
- A common usage is to pattern match on this very string...
- An attacker could thus change the control flow

Character escape functions

- `Char.escaped` is a function escaping some characters
- When called with a quote character, it returns the "\'" string
- So you can defeat the whole purpose of the mechanism with a one-liner

[OCaml] < yet strong 1/2

OCaml offers different encapsulation mechanisms[4]

```
module type Crypto = sig val id:int end;;

module C : Crypto  =
struct
  let id=Random.self_init(); Random.int 8192
  let key=Random.self_init(); Random.int 8192
end;;
```

It is a sealed box; the `id` value is visible while the `key` is hidden

`C.id` returns `- : int = 2570`

`C.key` throws `Error: Unbound value C.key`

[4] We consider here modules, given that the OCaml objects offer weaker guarantees

[OCaml] < yet strong 2/2

But this encapsulation can be bypassed (in earlier versions)

```
let rec oracle o1 o2 =
  let o = (o1 + o2)/2 in
  let module O = struct let id=C.id let key=o end in
  if (module O:Crypto)>(module C:Crypto)
  then oracle o1 o
  else (if (module O:Crypto)<(module C:Crypto)
        then oracle o o2
        else o);;

oracle 0 8192;;
```

This code would return the `key` value at runtime; we could not open the box, but we could use a weighing scale...

[OCaml] An early off-by-one

On a 32-bit machine, as many languages, OCaml does not handle integer overflow

```
# let x = 0x3fff_ffff;;
val x : int = 1073741823
# x+1;;
- : int = -1073741824
```

[OCaml] An early off-by-one

On a 32-bit machine, as many languages, OCaml does not handle integer overflow

```
# let x = 0x3fff_ffff;;
val x : int = 1073741823
# x+1;;
- : int = -1073741824
```

This is regrettable in such a cool language... but wait !

This x was $2^{30} - 1$ and not $2^{31} - 1$!

[OCaml] An early off-by-one

On a 32-bit machine, as many languages, OCaml does not handle integer overflow

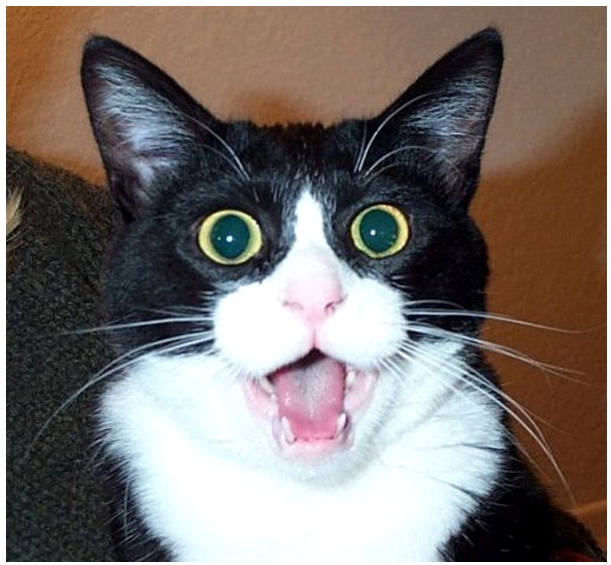
```
# let x = 0x3fff_ffff;;
val x : int = 1073741823
# x+1;;
- : int = -1073741824
```

This is regrettable in such a cool language... but wait !

This x was $2^{30} - 1$ and not $2^{31} - 1$!

Indeed, `int` represents a signed 31-bit integer in OCaml!

L'instant PHP



[Php] Internet et les vidéos de SHA

What is the relation with security?

```
$h1= md5 ('QNKCDZO');
$h2= md5 ('240610708');
$h3= md5 ('A169818202');
$h4= md5 ('aaaaaaaaaaaumdozb');
$h5= sha1('badthingsrealmlavznik');
```

Compared using ==, which one are equal?

A. None, of course	**C.** h1, h3 and h4
B. h3 and h5	**D.** Answer D

[Php] Internet et les vidéos de SHA

Answer D:

All of them!

In Php:

```
'0e830400451993494058024219903391' ==
'0e462097431906509019562988736854' ==
'0e590126417109547563244339779435' ==
'000e994639666666707280479226342 4' ==
'00e6350478108627283429100248932178194894'
```

[5] and phpBB in 2011 ...

[Php] Internet et les vidéos de SHA

Answer D:

All of them!

In Php:

```
'0e830400451993494058024219903391' ==
'0e462097431906509019562988736854' ==
'0e590126417109547563244339779435' ==
'000e994639666666707280479226342 4' ==
'00e635047810862728342910024893217 8194894'
```

Simple Machines Forum <= 2.0.3 Admin Password Reset (2013)[5]

```
if (empty($_POST['code']) ||
  substr($realCode, 0, 10) != substr(md5($_POST['code']), 0, 10))
```

[5] and phpBB in 2011 ...

Outline

Outline

[Java] *Clone Wars*

Extract of the official specification of the Java language, regarding the `clone` method of the `Object` class:

> *The general intent is that, for any object x, the expression:* `x.clone()!= x` *will be true, and that the expression:* `x.clone().getClass()== x.getClass()` *will be true, but these are not absolute requirements. While it is typically the case that:* `x.clone().equals(x)` *will be true, this is not an absolute requirement.*

The specification of the serialization operations (`writeObject` and `readObject`) is also quite puzzling

Outline

[C] *Cast-a-niet*

The compiler *could* help you

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    char *hello = "hello, world";

    hello[0] = 'Y';
    hello[1] = 'o';

    return 0;
}
```

[C] *Cast-a-niet*

The compiler *could* help you

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    char *hello = "hello, world";

    hello[0] = 'Y';
    hello[1] = 'o';

    return 0;
}
```

Program output is `Segmentation fault`. Error **is** predictable (cast from a constant byte array from RO section to a mutable array), but even with `-Wall -Wextra` there is no warning.

Outline

What now? (1/2)

How we choose a language:

- generally, we use what we know
- otherwise it's performance

What now? (1/2)

How we choose a language:

- generally, we use what we know
- otherwise it's performance

How we should choose a language:

- what you want to do: parsing, low-level programing, GUI, ...
- ideally: **compromise** between language **security properties**, **knowledge**, and **performance**
- performance cannot justify everything!

Security cannot rely on developers only

What now? (2/2)

Tools can help you

- always ask for all warnings
  (`-Wall -Wextra -Wwrite-strings -Wconversions ...`)
- **never** do quick 'n dirty
- more time thinking, less time debugging

Good habits can help you

- always test results
- use whitelists, not blacklists
- KISS
- do not use all features of a language, nor write ASCII art (IOCCC contest / most Rust programs)

A Word on Polyglots

Idea: use multiple languages in a project

Pros

- write parsers in a safe language
- write low-level and networking stuff in an efficient language
- better architecture

Cons

- more complex
- serialization is dangerous
- more problems: `ctypes` is even more dangerous
- now you have the problems of several languages!

Lessons learned

- Programming languages can impact software security
- There is room for improvement in them
- We could benefit from more research and tools

- Writing secure software requires a broad vision in many aspects of computer science
- Teaching should take more those aspects into account

- Our job is safe!

Questions?

Thank you for your attention

first.last@ssi.gouv.fr