Mind your Language(s)!

Langages de développement et sécurité

É. Jaeger & O. Levillain & P. Chifflier, ANSSI

RESSI, 22 mai 2015

An unreliable programming language generating unreliable programs constitutes a far greatest risk to our environment and to our society than unsafe cars, toxic pesticides, or accidents at nuclear power stations. Be vigilant to reduce that risk, not to increase it.

C.A.R. Hoare

Motivations

En 2005, un industriel interroge la DCSSI pour savoir si le langage JAVA peut être utilisé développer un produit de sécurité

Cette question, généralisée, a mené à différentes études dont

- JAVASEC : sécurité du langage JAVA
- LAFOSEC : sécurité des langages fonctionnels (dont OCAML)

L'une des leçons de ces études, c'est que les questions de l'ANSSI à propos des langages ne sont pas toujours partagées ou comprises

Sécurité ?

Spécification de deux fonctions pour la compression (Zip) et la décompression (Unzip) de fichiers

Fonctionnel

- $\forall\,(f : \mathrm{File}),\ \mathrm{Unzip}(\mathrm{Zip}\,f) = f$

Sécurité

- $\forall\,(c : \mathrm{File}),\ (\neg\exists\,(f : \mathrm{File}),\ \mathrm{Zip}\,f = c) \Rightarrow \mathrm{Unzip}\,c = \mathrm{Error}$
- En particulier, ne pas avoir confiance en un champ annonçant à l'avance la taille du fichier décompressé

Sécurité et langages

Quelques aspects intéressants d'un langage en termes de sécurité

- Faux amis ou pièges
- Constructions non spécifiées ou non définies mais disponibles
- Possibilités d'offuscation pour un développeur malicieux
- Bugs ou comportements inappropriés des outils de développement
- Limitations des capacités d'analyse
- Surprises à l'exécution...

Ce qui est signalé dans la suite n'est pas forcément une erreur, mais « attire l'attention » des experts en sécurité

Sécurité et langages

Quelques aspects intéressants d'un langage en termes de sécurité

- Faux amis ou pièges
- Constructions non spécifiées ou non définies mais disponibles
- Possibilités d'offuscation pour un développeur malicieux
- Bugs ou comportements inappropriés des outils de développement
- Limitations des capacités d'analyse
- Surprises à l'exécution...

Ce qui est signalé dans la suite n'est pas forcément une erreur, mais « attire l'attention » des experts en sécurité

- L'objet n'est pas la critique d'un langage en particulier
- Aucun langage n'a été endommagé pendant cette étude

Plan

Tutoriel 1 : de l'importance de (la vie) privée

[Java] Encore une objection

L'encapsulation objet est-elle un mécanisme de sécurité ?

Source (java/Introspect.java)

```
import java.lang.reflect.*;
class Secret { private int x = 42; }
public class Introspect {
  public static void main (String[] args) {
    try { Secret o = new Secret();
          Class c = o.getClass();
          Field f = c.getDeclaredField("x");
          f.setAccessible(true);
          System.out.println("x="+f.getInt(o));
        }
    catch (Exception e) { System.out.println(e); }
  }
}
```

L'introspection peut être interdite dans la politique de sécurité Java, mais c'est complexe et des effets secondaires sont probables

[Java] Nombrilisme

Java permet de définir des classes internes

Source (java/Innerclass.java)

```
public class Innerclass {
  private static int a=42;

  static public class Innerinner {
    private static int b=54;
    public static void print() {
      System.out.println(Innerclass.a);
    }
  }

  public static void main (String[] args) {
    System.out.println(Innerinner.b);
    Innerinner.print();
  }
}
```

Pourtant les classes internes ne sont pas supportées en *bytecode* !

Tutoriel 2 : l'égalité, une notion complexe

[JavaScript] Certains sont plus égaux que d'autres

JavaScript offre également tout le confort moderne...

Source (js/unification2.js)

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

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

[JavaScript] Certains sont plus égaux que d'autres

JavaScript offre également tout le confort moderne...

Source (js/unification2.js)

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

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

L'affichage obtenu est `Equal` puis `Different`

[Java] Encore une égalité contrariante

Au moins avec l'égalité physique, on sait à quoi s'en tenir. . . sauf en cas d'interactions subtiles avec des bibliothèques standards innovantes

Source (java/IntegerBoxing.java)

```
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] Encore une égalité contrariante

Au moins avec l'égalité physique, on sait à quoi s'en tenir. . . sauf en cas d'interactions subtiles avec des bibliothèques standards innovantes

Source (java/IntegerBoxing.java)

```
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");
```

`a1==a2`, mais pas de second affichage, qui veut jouer aux devinettes ?

[Ruby] Ordres, contre-ordres

La surcharge permet de spécialiser pour adopter le comportement qui semble le plus « naturel », même si ce n'est pas toujours cohérent...

Source (ruby/intervals.rb)

```
> "a"<"ab"
=> true

> "ab"<"b"
=> true

> ("a".."b")==="ab"
=> true

> ("a".."b").each { |x| print (x.to_s+".") }
a.b. => "a".."b"
```

Localement, tout semble avoir du sens, mais globalement, la sémantique des intervalles de chaînes de caractères n'est pas du tout claire

[JavaScript] Reconversion

Faut-il préférer *cast* et surcharge, ou associativité et transitivité ?

[JavaScript] Reconversion

Faut-il préférer *cast* et surcharge, ou associativité et transitivité ?

En JavaScript, on a `'0'==0` qui est vrai, de même `0=='0.0'`, par contre `'0'=='0.0'` est faux ; en d'autres termes, l'égalité n'est pas transitive

[JavaScript] Reconversion

Faut-il préférer *cast* et surcharge, ou associativité et transitivité ?

En JavaScript, on a `'0'==0` qui est vrai, de même `0=='0.0'`, par contre `'0'=='0.0'` est faux ; en d'autres termes, l'égalité n'est pas transitive

Autre exemple avec l'opérateur `+`, qui peut être l'addition d'entiers ou la concaténation de chaînes, mais qui est dans les deux cas associatif

Source (js/cast3.js)

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

[JavaScript] Reconversion

Faut-il préférer *cast* et surcharge, ou associativité et transitivité ?

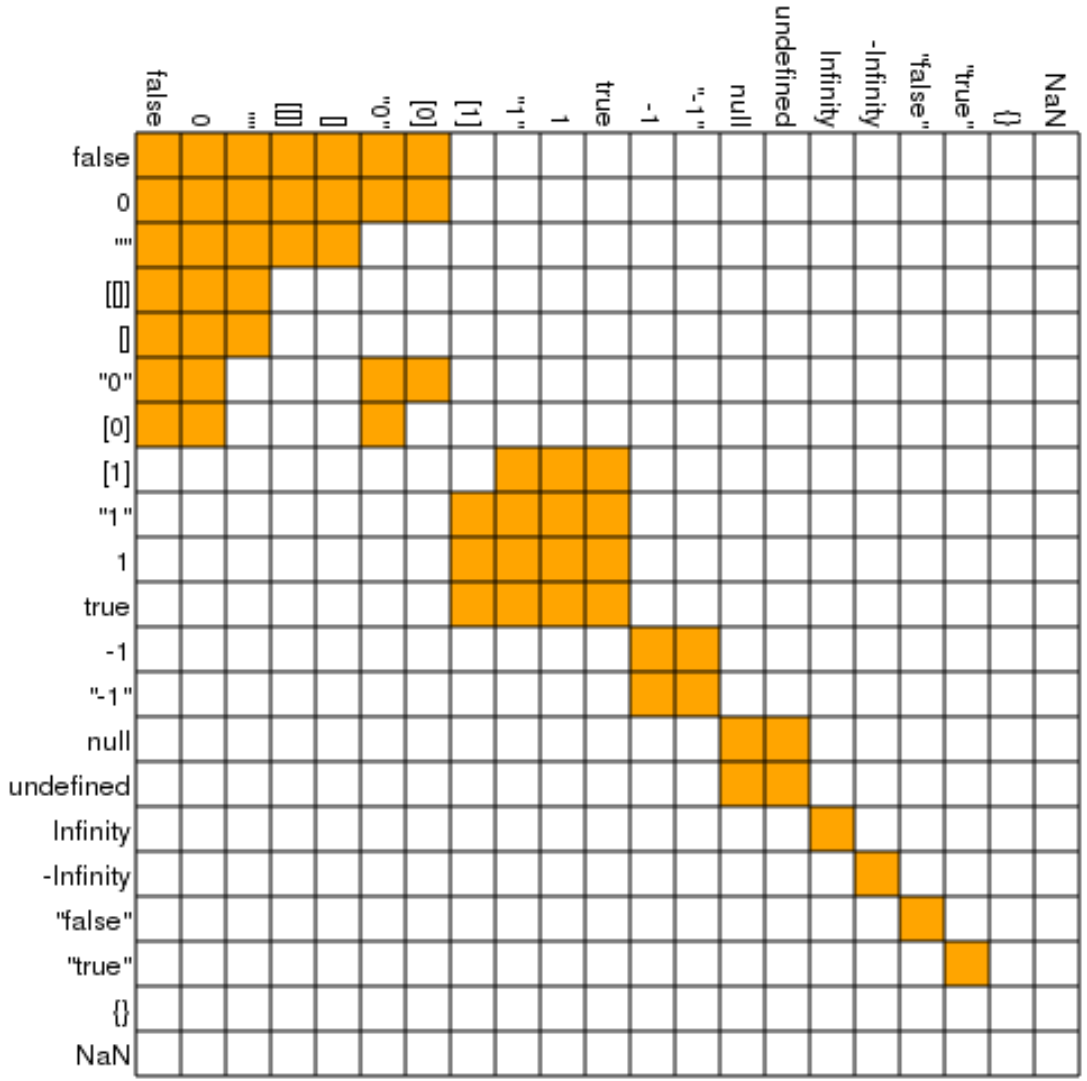
En JavaScript, on a `'0'==0` qui est vrai, de même `0=='0.0'`, par contre `'0'=='0.0'` est faux ; en d'autres termes, l'égalité n'est pas transitive

Autre exemple avec l'opérateur `+`, qui peut être l'addition d'entiers ou la concaténation de chaînes, mais qui est dans les deux cas associatif

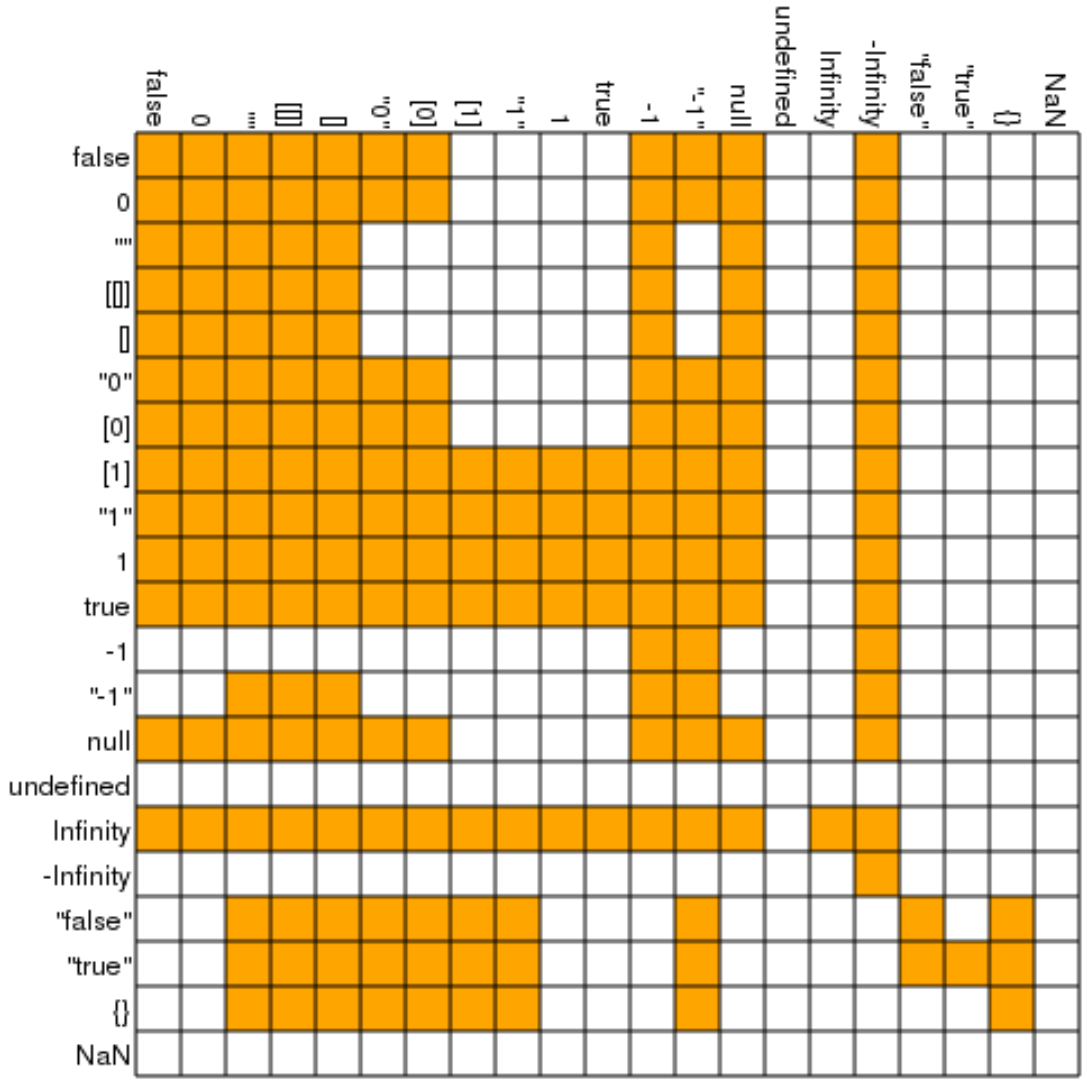
Source (js/cast3.js)

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

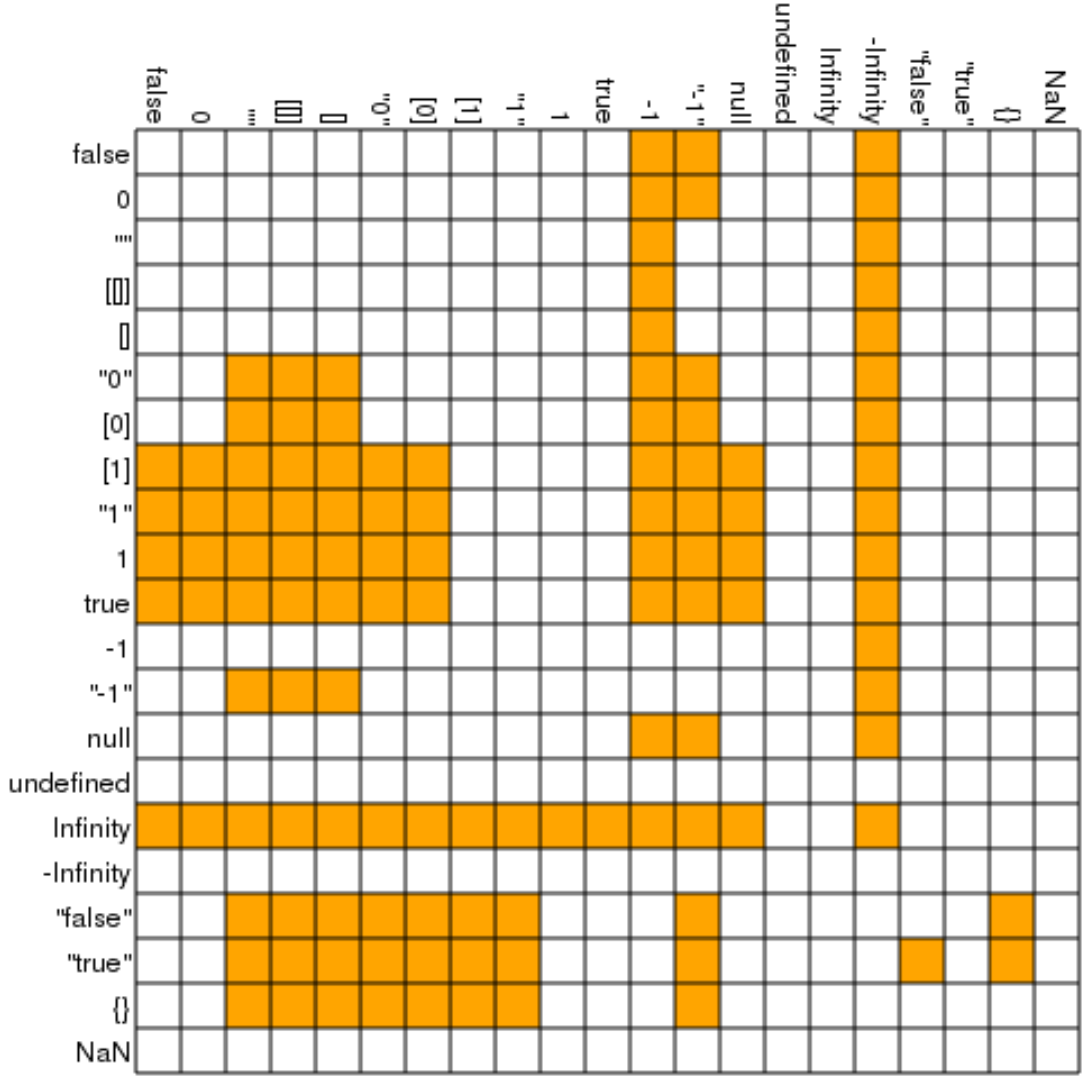
`3Foo`, `Foo12` et `Foo3`


false
0
""
[[]]
[]
"0"
[0]
[1]
"1"
1
true
-1
"-1"
null
undefined
Infinity
-Infinity
"false"
"true"
{}
NaN

Égal ==

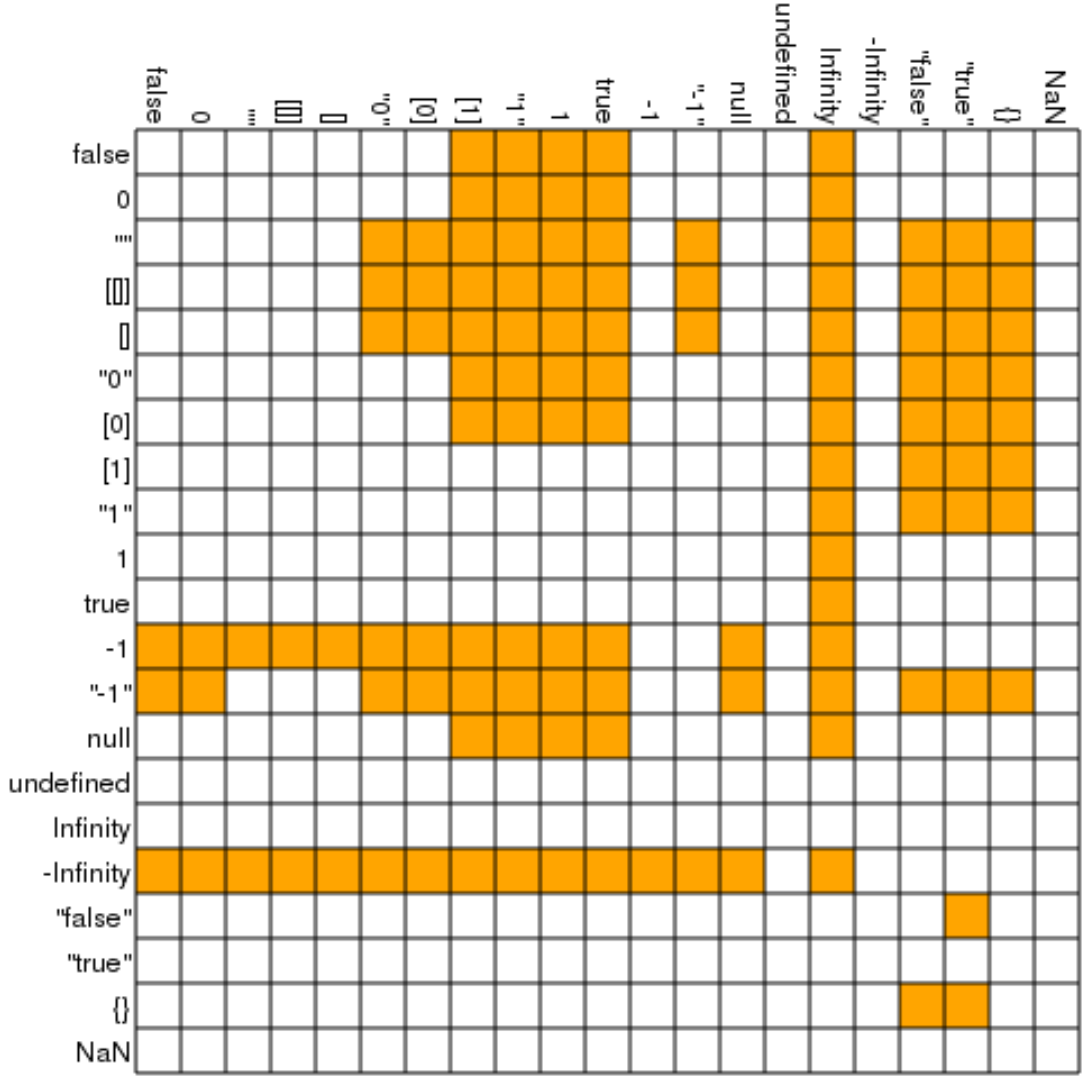


Plus petit ou égal <=


false
0
""
[[]]
[]
"0"
[0]
[1]
"1"
1
true
-1
"-1"
null
undefined
Infinity
-Infinity
"false"
"true"
{}
NaN

Plus petit <

[JavaScript] Enter the Matrix



Plus grand >

Source (php/castincr.php)

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

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

if ("0xF9"=="249") { print("Equal\n"); }
else { print("Different\n"); }
```

Source (php/castincr.php)

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

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

if ("0xF9"=="249") { print("Equal\n"); }
else { print("Different\n"); }
```

La première ligne affiche 3 (entier)

Source (php/castincr.php)

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

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

if ("0xF9"=="249") { print("Equal\n"); }
else { print("Different\n"); }
```

La première ligne affiche `3` (entier)

La seconde ligne affiche `2d9` (chaîne), `2e0` (chaîne) puis `3` (flottant)

Source (php/castincr.php)

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

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

if ("0xF9"=="249") { print("Equal\n"); }
else { print("Different\n"); }
```

La première ligne affiche `3` (entier)

La seconde ligne affiche `2d9` (chaîne), `2e0` (chaîne) puis `3` (flottant)

La troisième ligne affiche `Equal`

Quel rapport avec la sécurité ? Voici un exemple

Source (php/hash.php)

```
$s1='QNKCDZO'; $h1=md5($s1);
$s2='240610708'; $h2=md5($s2);
$s3='A169818202'; $h3=md5($s3);
$s4='aaaaaaaaaaaumdozb'; $h4=md5($s4);
$s5='badthingsrealmlavznik'; $h5=sha1($s5);

if ($h1==$h2) print("Collision\n");
if ($h2==$h3) print("Collision\n");
if ($h3==$h4) print("Collision\n");
if ($h4==$h5) print("Collision\n");
```

Quel rapport avec la sécurité ? Voici un exemple

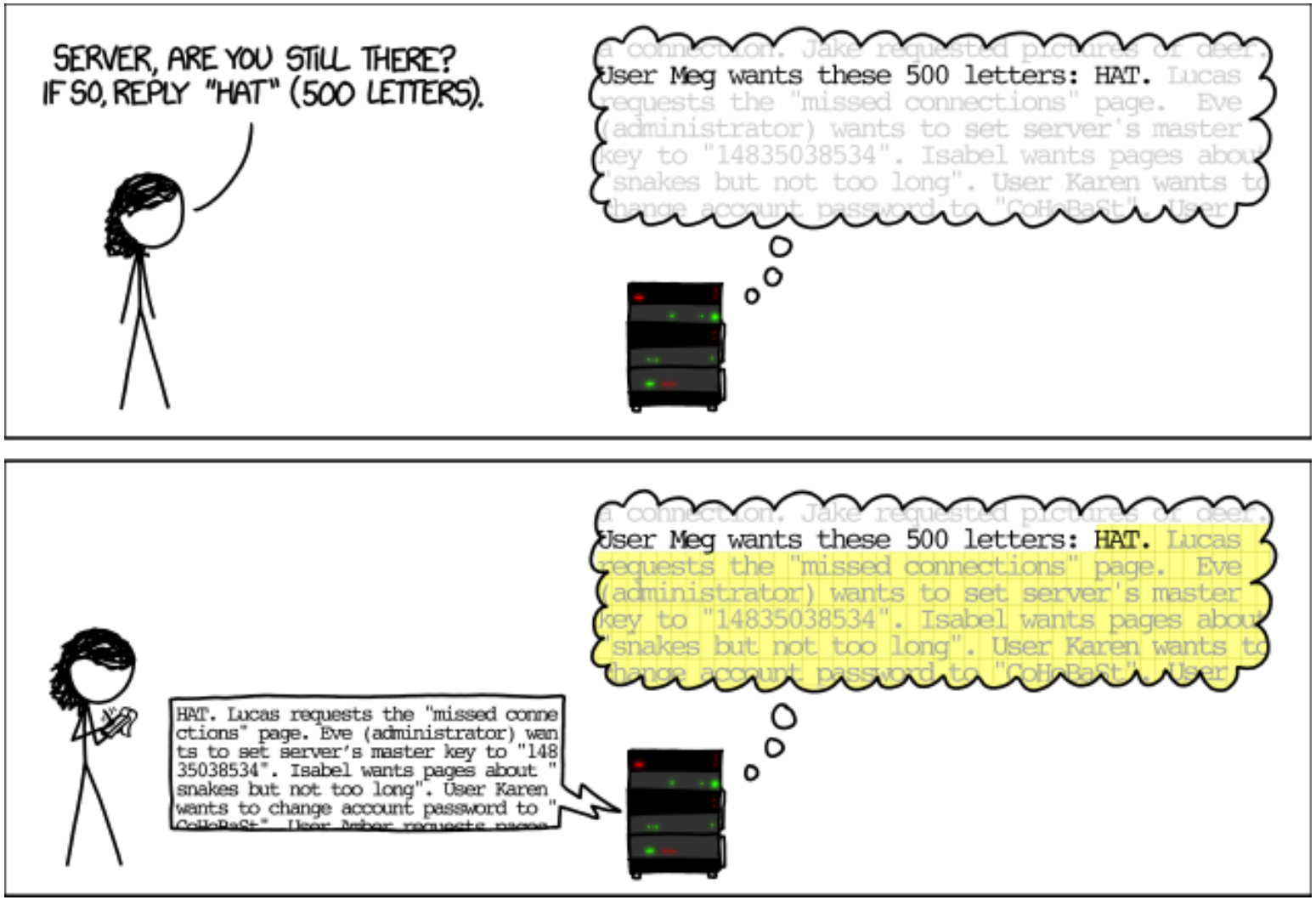
Source (php/hash.php)

```
$s1='QNKCDZO'; $h1=md5($s1);
$s2='240610708'; $h2=md5($s2);
$s3='A169818202'; $h3=md5($s3);
$s4='aaaaaaaaaaaumdozb'; $h4=md5($s4);
$s5='badthingsrealmlavznik'; $h5=sha1($s5);

if ($h1==$h2) print("Collision\n");
if ($h2==$h3) print("Collision\n");
if ($h3==$h4) print("Collision\n");
if ($h4==$h5) print("Collision\n");
```

`Collision` est affiché 4 fois, mais ne concluez pas trop vite que MD5 et SHA1 sont mis en cause ici

Table ronde 1 : le code ouvert est gage de sécurité


SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHeRaSt". User
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "

La faille *Heartbleed* (CVE-2014-160) a été révélée en avril 2014

Concrètement, un serveur HTTPS sur deux de la planète était concerné avec une compromission possible

- des clés privées
- des mots de passe
- de toute autre information présente en mémoire du process. . .

L'intégration de ce service de sécurité cryptographique a induit une vulnérabilité dont l'impact va très au-delà du périmètre de ce service

C'est un simple oubli de vérification de bornes dans le code d'une fonction non critique du protocole SSL/TLS

La faille *Heartbleed* (CVE-2014-160) a été révélée en avril 2014

Concrètement, un serveur HTTPS sur deux de la planète était concerné avec une compromission possible

- des clés privées
- des mots de passe
- de toute autre information présente en mémoire du process...

L'intégration de ce service de sécurité cryptographique a induit une vulnérabilité dont l'impact va très au-delà du périmètre de ce service

C'est un simple oubli de vérification de bornes dans le code d'une fonction non critique du protocole SSL/TLS

Il est temps de se rappeler où l'on trouve des serveur HTTPS...

Source (shell/login.sh)

```
#!/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
```

Source (shell/login.sh)

```
#!/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
```

Un mauvais code PIN sera rejeté ; par contre, si l'utilisateur saisit des caractères non numériques, l'accès lui sera accordé

[C] True, False, FILE_NOT_FOUND

Focus sur la vulnérabilité *Goto Fail* de GnuTLS en mars 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] Epic Apple's Goto Fail

Encore un bug dans une bibliothèque cryptographique révélé en 2014

Source (c/gotofail.c)

```
/* 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;
```

La syntaxe n'aide pas, mais le compilateur ne semble pas non plus se préoccuper de signaler du code manifestement mort...

[C] Syntaxe, priez pour nous

Une modification du noyau Linux [1]

Source (c/kernel.diff)

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

1. Cf. lwn.net/Articles/57135/

[C] Syntaxe, priez pour nous

Une modification du noyau Linux [1]

Source (c/kernel.diff)

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

Piégeage pur et simple : lorsque la condition sur `options` est vraie, `current->uid` devient `0` (*i.e.* le process passe `root`)

L'attaquant joue sur la confusion entre `=` et `==`, mais aussi le fait que l'affectation renvoie une valeur, que le typage ne distingue pas un booléen d'un entier, que le et booléen est paresseux, *etc.*

1. Cf. `lwn.net/Articles/57135/`

Pot pourri

[Perl] The Perl Jam (31c3)

Source (perl/perljam1.pl)

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

`http://index.cgi?user=user'` fonctionne comme prévu

[Perl] The Perl Jam (31c3)

Source (perl/perljam1.pl)

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

`http://index.cgi?user=user'` fonctionne comme prévu

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

[Perl] The Perl Jam (31c3)

Source (perl/perljam1.pl)

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

`http://index.cgi?user=user'` fonctionne comme prévu

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

Source (perl/perljam2.pl)

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

[Python] Localité en fête

Python offre une construction syntaxique proche du `map` classique sur les listes, la définition de listes en compréhension

Source (python/listcomp.py)

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

Que se passe-t-il si ensuite on tape `s` à l'invite ?

[Python] Localité en fête

Python offre une construction syntaxique proche du `map` classique sur les listes, la définition de listes en compréhension

Source (python/listcomp.py)

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

Que se passe-t-il si ensuite on tape `s` à l'invite ?

À moins d'utiliser la dernière version de Python 3, `s` vaut `3`, alors que la variable `s` devrait être locale (liée)

En OCaml le code est statique et les chaînes sont mutables ; mais qu'en est-il des chaînes apparaissant dans le code ?

Source (ocaml/mutable.ml)

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

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

check true;;
check false;;
```

En OCaml le code est statique et les chaînes sont mutables ; mais qu'en est-il des chaînes apparaissant dans le code ?

Source (ocaml/mutable.ml)

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

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

check true;;
check false;;
```

Les deux applications de `check` renvoient `"OK"`

L'exemple précédent n'est pas une redéfinition de la fonction `alert` mais un simple effet de bord ; pour s'en convaincre, voici ce que cela donne avec une fonction de la bibliothèque standard

Source (ocaml/mutablebool.ml)

```
let t=string_of_bool true in
  t.[0]<-'f'; t.[1]<-'a'; t.[3]<-'x';;

Printf.printf "1=1 est %b\n" (1=1);;
```

Le code affiche `1=1 est faux` ; d'autres fonctions intéressantes sont concernées, par exemple `Char.escaped` (!) ainsi que certains *patterns* de développement usuels basés sur les exceptions

Plan

[Java] Clone Wars

Extrait de la spécification officielle du langage Java relative à la méthode `clone` de la classe `Object`

> *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.*

La spécification des opérations de sérialisation (`writeObject` et `readObject`) est aussi assez intriguante

[C] L'auberge espagnole

Un extrait de "*The* C *programming language (Second edition)*" de *B. W. Kernighan & D. M. Ritchie*

> *The direction of truncation for* `/` *and the sign of the result for* `%` *are machine-dependent for negative operands, as is the action taken on overflow or underflow.*

La question est de savoir comment tester la conformité d'un compilateur à cette spécification non-déterministe. . . Pensez-y

[C] L'auberge espagnole

Un extrait de "*The C programming language (Second edition)*" de *B. W. Kernighan & D. M. Ritchie*

> *The direction of truncation for* `/` *and the sign of the result for* `%` *are machine-dependent for negative operands, as is the action taken on overflow or underflow.*

La question est de savoir comment tester la conformité d'un compilateur à cette spécification non-déterministe. . . Pensez-y

Votre test rejeterait-il un compilateur changeant l'arrondi à chaque appel, ce qui permettrait d'avoir `1/-2==1/-2` évalué à faux ? C'est une instanciation de ce qu'on appelle le *Paradoxe du raffinement*, résolu avec le standard C99

Plan

À propos du développement

À quoi devrait être attentif un développeur ?

- Les langages ne sont pas vos amis : coder sécurisé, c'est comme jongler avec des tronçonneuses
- Sélectionner les « bons » langages (ou les *frameworks*)
  - Est-il raisonnable de faire de la cryptographie en JavaScript ?
  - À défaut utiliser les bons outils et les bonnes options
- Maîtriser chaque langage utilisé
  - Faux amis, mots clés communs avec des sens différents
  - Robustesse des mécanismes proposés
  - Ne pas abuser des gadgets (faut-il laisser l'ordinateur deviner ?)
- Attention aux traits dynamiques : évaluations, liaisons tardives, désérialisation, *etc.* peuvent devenir source de vulnérabilités
- Ne pas forcer son talent. . . et penser au-delà du fonctionnel !
- Se méfier des codes tiers

À propos de l'enseignement

Comment former un développeur ou un évaluateur ?

- La sécurité n'est pas un module qui s'intègre parmi d'autres
  - Elle ne peut pas être totalement déléguée aux "experts"
  - Que devrait savoir tout développeur à propos de la sécurité ?
- Savoir aller au-delà du fonctionnel (un plus court chemin)
  - L'attaquant cherche les erreurs, préconditions et valeurs observables mais pourtant hors modèle, *etc.*
  - Le développeur de sécurité doit identifier et écarter tout ce qui peut mal tourner (conserver un seul chemin acceptable)
- Maîtriser les fondamentaux
  - Sémantique des langages
  - Théorie de la compilation
  - Principes des systèmes d'exploitation
  - Architecture des ordinateurs
  - ...

À propos des langages

Comment aider à l'amélioration de la sécurité ou l'assurance ?

- La spécification d'un langage est idéalement complète, déterministe et non ambiguë (voire formalisée)
- *Simple is beautiful*
  - Ne conserver que ce qui est nécessaire
  - Éviter ce qui est complexe ou dénué de sens
  - Ne pas contrarier l'intuition ou la logique élémentaire
- Sans maîtrise, la puissance n'est rien
  - Faciliter la lisibilité et la traçabilité : un mot clé pour un concept, des notations cohérentes, *etc.*
  - Ne pas confondre aide au développeur avec laxisme ou devinettes
  - Introspection, évaluation, traits dynamiques rendent toute forme d'analyse délicate voire impossible

- Projet SecurOCaml (FUI 18)

À propos des outils

Quels outils (ou options) pour la sécurité et l'assurance ?

- Ce qui n'est pas spécifié pour le langage devrait être interdit par les outils – ou au moins signalé
- Implémenter les vérifications possibles, et les faire au plus tôt
- Minimiser les manipulations silencieuses
- Savoir aller au-delà du fonctionnel
  - Le raisonnement de sécurité nécessite de penser au-delà des interfaces d'une boîte noire
  - Certaines optimisations sont inappropriées en sécurité
- Étendre le domaine des invariants de compilation
  - Modèle mémoire reflétant l'encapsulation
  - Préserver le flot d'exécution même en présence de fautes
- Disposer d'outils maîtrisés voire de confiance

Remerciements

Les exemples de cette présentation sont fournis ou inspirés par :

- Les laboratoires de l'ANSSI
- Les participants à l'étude JavaSec
- Les participants à l'étude LaFoSec
- Différents sites et blogs, notamment :
  - `www.thedailywtf.com`, `www.xkcd.com`
  - le site de la société MLState
  - *Sami Koivu* (*Slightly Random Broken Thoughts*)
  - *Jeff Atwood* (*Coding Horror*)
  - *Software Engineering Not At School*
  - *Functional Orbitz*
  - *Rob Kendrick* (*Some dark corners of* C)

Sans oublier, bien entendu, l'aimable collaboration des concepteurs des langages et des outils ;-)

If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.

Gerald M. Weinberg

Software and cathedrals are very much the same – first we build them, then we pray.

Sam Redwine

Beware of bugs in the above code ; I have only proved it correct, not tried it.

Donald Knuth