Mind your Language(s)!

Olivier Levillain

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

The tools we are trying to use and the language or notations we are using to express or record our thoughts, are the major factor determining what we can think or express at all !

Edsger W. Dijkstra

Don't you see that the whole aim of Newspeak is to narrow the range of thought ? In the end we shall make thought-crime literally impossible, because there will be no words in which to express it.

1984, George Orwell

Sécurité et langages

Quelques aspects intéressants d'un langage en termes de sécurité

- 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'objectif ici n'est *pas* de critiquer un langage en particulier

Historique

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

Les exemples présentés ici sont issus de travaux menés dans les laboratoires de l'ANSSI, initiés par Éric Jaeger [1]

1. La plupart des jeux de mots sont de lui...

Plan

Illustrations

Réflexions et exercices

Plan

Plan

[C] Questions stratégiques

Si on joue avec les effets de bord, la stratégie d'évaluation devient significative, et des subtilités peuvent apparaître

Source (c/effect.c)

```
{ int c=0; printf("%d %d\n",c++,c++); }

{ int c=0; printf("%d %d\n",++c,++c); }

{ int c=0; printf("%d %d\n",c=1,c=2); }
```

[C] Questions stratégiques

Si on joue avec les effets de bord, la stratégie d'évaluation devient significative, et des subtilités peuvent apparaître

Source (c/effect.c)

```
{ int c=0; printf("%d %d\n",c++,c++); }

{ int c=0; printf("%d %d\n",++c,++c); }

{ int c=0; printf("%d %d\n",c=1,c=2); }
```

La première ligne affiche `1 0`

[C] Questions stratégiques

Si on joue avec les effets de bord, la stratégie d'évaluation devient significative, et des subtilités peuvent apparaître

Source (c/effect.c)

```
{ int c=0; printf("%d %d\n",c++,c++); }

{ int c=0; printf("%d %d\n",++c,++c); }

{ int c=0; printf("%d %d\n",c=1,c=2); }
```

La première ligne affiche `1 0`
La seconde ligne affiche `2 2`

[C] Questions stratégiques

Si on joue avec les effets de bord, la stratégie d'évaluation devient significative, et des subtilités peuvent apparaître

Source (c/effect.c)

```
{ int c=0; printf("%d %d\n",c++,c++); }

{ int c=0; printf("%d %d\n",++c,++c); }

{ int c=0; printf("%d %d\n",c=1,c=2); }
```

La première ligne affiche `1 0`
La seconde ligne affiche `2 2`
La troisième ligne affiche `1 1`

[C] La goutte d'eau

Petit rappel sur le principe des *Buffer Overflows*[2]

Source (c/overflow.c)

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

void set(int s,int v) { *(&s-s)=v; }

void bad() { printf("Bad things happen!\n"); exit(0); }

int main(void) {
  set(1,(int)bad); printf("Hello world\n"); return 0;
}
```

2. Ici dans une version balèze, car sans *Buffer*

[C] La goutte d'eau

Petit rappel sur le principe des *Buffer Overflows*[2]

Source (c/overflow.c)

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

void set(int s,int v) { *(&s-s)=v; }

void bad() { printf("Bad things happen!\n"); exit(0); }

int main(void) {
  set(1,(int)bad); printf("Hello world\n"); return 0;
}
```

`Bad things happen!`
La pile est corrompue, on peut surcharger des variables, modifier une adresse de retour voire injecter du code...

2. Ici dans une version balèze, car sans *Buffer*

[C] Faire mauvaise impression

Mais les manipulations de piles sont parfois bien cachées. . .

Source (c/stringformat3.c)

```
#include <stdio.h>

char *f="%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.\
%08x.%08x.%08x.%08x.%08x.%08x.%n";

void strfmtattack() { printf(f); }

int main(void) {
  int s=0x12345;
  int *p=&s;
  strfmtattack();
  if (s!=0x12345) printf("Bad things happen! s=%08x\n",s);
  return 0;
}
```

[C] Faire mauvaise impression

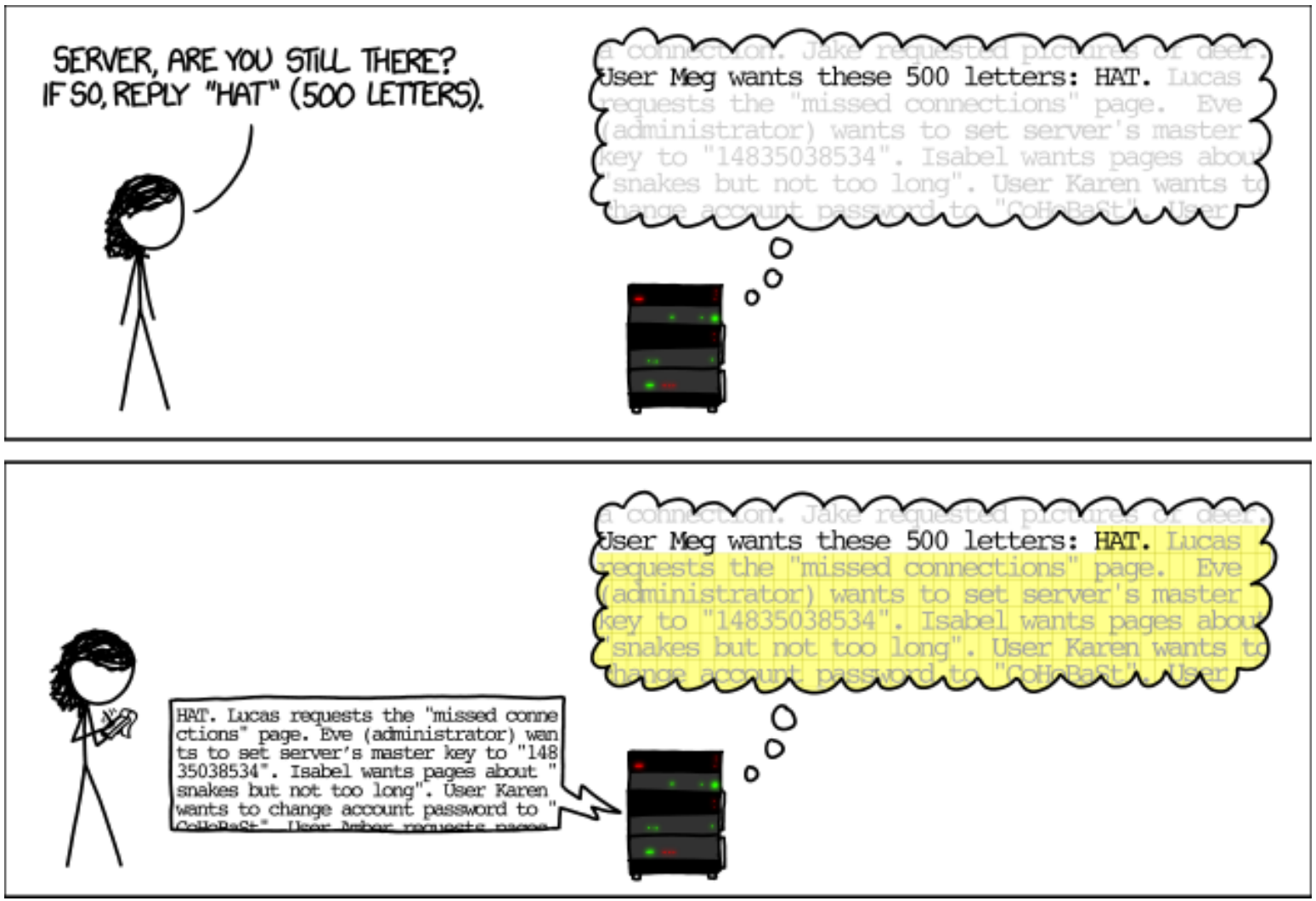
Mais les manipulations de piles sont parfois bien cachées. . .

Source (c/stringformat3.c)

```
#include <stdio.h>

char *f="%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.\
%08x.%08x.%08x.%08x.%08x.%08x.%n";

void strfmtattack() { printf(f); }

int main(void) {
  int s=0x12345;
  int *p=&s;
  strfmtattack();
  if (s!=0x12345) printf("Bad things happen! s=%08x\n",s);
  return 0;
}
```

```
...0036b225.00fdd280.00000000.00012345.Bad things happen! s=0000007e
```

[C] Écho-logie 1/2



[C] Écho-logie 2/2

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

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

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

L'intégration d'un service de sécurité (cryptographique) 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

[C] Système de *cast* 1/2

Le compilateur C cherche également à rendre service, que vaut `z` ?

Source (c/cast0.c)

```
{ int x=3; int y=4; float z=x/y; }
```

[C] Système de *cast* 1/2

Le compilateur C cherche également à rendre service, que vaut `z` ?

Source (c/cast0.c)

```
{ int x=3; int y=4; float z=x/y; }
```

```
0.0
```

[C] Système de *cast* 1/2

Le compilateur C cherche également à rendre service, que vaut `z` ?

Source (c/cast0.c)

```
{ int x=3; int y=4; float z=x/y; }
```

```
0.0
```

Source (c/cast1.c)

```
{ unsigned char x = 128; unsigned char y = 2;
  unsigned char z = (x * y) / y; }
```

[C] Système de *cast* 1/2

Le compilateur C cherche également à rendre service, que vaut `z` ?

Source (c/cast0.c)

```
{ int x=3; int y=4; float z=x/y; }
```

```
0.0
```

Source (c/cast1.c)

```
{ unsigned char x = 128; unsigned char y = 2;
  unsigned char z = (x * y) / y; }
```

128 (ce n'est pas une optimisation du compilateur mais un *cast*)

[C] Système de *cast* 2/2

Pour les curieux, voici le code assembleur associé à `cast1.c`

Source (c/castregister.asm)

```
    unsigned char x = 128;
          movb    $-128, -1(%rbp)
    unsigned char y = 2;
          movb    $2, -2(%rbp)
    unsigned char z = (x * y) / y;
          movzbl  -1(%rbp), %edx
          movzbl  -2(%rbp), %eax
          imull   %edx, %eax
          movzbl  -2(%rbp), %edi
          cltd
          idivl   %edi
          movb     %al, -3(%rbp)
```

[C] Complément d'information

Avant de présenter les exemples suivants, un petit rappel : combien y a-t-il de solutions pour l'équation $x = -x$ en C ?

Source (c/cast2.c)

```
#include <stdio.h>

int main(void) {
  int z=0;
  if (z==-z) printf("%d==-(%d)\n",z,z);

  int r=1<<((sizeof(int)*8)-1);
  if (r==-r) printf("%d==-(%d)\n",r,r);

  return 0;
}
```

`0==-(0)` et `-2147483648==-(-2147483648)`, c'est ce qui fait tout le charme du complément à 2 pour représenter les entiers signés

[C] C'est mauvais signe...

Même quand on pense avoir bien compris, il reste des surprises...

Source (c/castsigned2.c)

```
{ unsigned char a = 1; signed char b = -1;
  if (a<b) printf("%d<%d\n",a,b);
  else printf("%d>=%d\n",a,b); }

{ unsigned int a = 1; signed int b = -1;
  if (a<b) printf("%d<%d\n",a,b);
  else printf("%d>=%d\n",a,b); }
```

[C] C'est mauvais signe...

Même quand on pense avoir bien compris, il reste des surprises...

Source (c/castsigned2.c)

```
{ unsigned char a = 1; signed char b = -1;
  if (a<b) printf("%d<%d\n",a,b);
  else printf("%d>=%d\n",a,b); }

{ unsigned int a = 1; signed int b = -1;
  if (a<b) printf("%d<%d\n",a,b);
  else printf("%d>=%d\n",a,b); }
```

Avec les types `char`, on obtient `1>=-1`, alors qu'avec les types `int` on obtient `1<-1` – encore des subtilités sur les promotions

[C] Compromission sans condition

Une modification du noyau LINUX [3]

Source (c/kernel.diff)

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

3. Cf. lwn.net/Articles/57135/

[C] Compromission sans condition

Une modification du noyau Linux [3]

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.*

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

[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. . .

Plan

[Java] Charge `static` 1/2

Le comportement des programmes objet très simples est parfois surprenant : que fait le code suivant lorsque `Mathf.pi=3.1415` ?

Source (java/StaticInit.java)

```
class StaticInit {
  public static void main(String[] args) {
    if (Mathf.pi-3.1415<0.0001)
      System.out.println("Hello world");
    else
      System.out.println("Hello strange universe");
  }
}
```

[Java] Charge `static` 1/2

Le comportement des programmes objet très simples est parfois surprenant : que fait le code suivant lorsque `Mathf.pi=3.1415` ?

Source (java/StaticInit.java)

```
class StaticInit {
  public static void main(String[] args) {
    if (Mathf.pi-3.1415<0.0001)
      System.out.println("Hello world");
    else
      System.out.println("Hello strange universe");
  }
}
```

À l'exécution, on obtient `Bad things happen!`

[Java] Charge `static` 2/2

L'explication du comportement de `StaticInit` se trouve dans `Mathf`

Source (java/Mathf.java)

```
class Mathf {
  static double pi=3.1415;
  static { // Do whatever you want here
           System.out.println("Bad things happen!");
           // Do not return to calling class
           System.exit(0); }
}
```

En Java le chargement d'une classe exécute le code d'initialisation de classe, même en l'absence d'appel à une méthode ou à un constructeur

[Java] *Serial killer*

Évitons donc toute référence à une classe externe non maîtrisée

Source (java/Deserial.java)

```
import java.io.*;
class Friend { } // Unlikely to be dangerous!
class Deserial {
  public static void main (String[] args)
    throws FileNotFoundException, IOException,
           ClassNotFoundException {
    FileInputStream fis = new FileInputStream("friend");
    ObjectInputStream ois = new ObjectInputStream(fis);
    Friend f=(Friend)ois.readObject();
    System.out.println("Hello world");
  }
}
```

[Java] *Serial killer*

Évitons donc toute référence à une classe externe non maîtrisée

Source (java/Deserial.java)

```
import java.io.*;
class Friend { } // Unlikely to be dangerous!
class Deserial {
  public static void main (String[] args)
    throws FileNotFoundException, IOException,
           ClassNotFoundException {
    FileInputStream fis = new FileInputStream("friend");
    ObjectInputStream ois = new ObjectInputStream(fis);
    Friend f=(Friend)ois.readObject();
    System.out.println("Hello world");
  }
}
```

Hélas à l'exécution on a `Bad things happen!` Le fichier sérialisé contient la classe de l'objet, mais lorsque le *cast* échoue c'est déjà trop tard

[Java] Sudonomicon

En guide d'introduction, un petit rappel...

Source (java/PrivArrayAccess.java)

```
class PrivArray {
  static private int[] tab={1,2};
  static public int[] get() { return tab; } // No set!
  static public void print()
    { System.out.println(tab[0]+","+tab[1]); }
}

class PrivArrayAccess {
  public static void main (String[] args) {
    PrivArray.print();
    PrivArray.get()[1]=3; // Do you set?
    PrivArray.print();
  }
}
```

Le programme affiche `1,2` puis `1,3`...

[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] 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 ?

[Java] Surcharge (pond des râles)

Le développeur peut parfois mettre son propre grain de sel

Source (java/Confuser.java)

```
class Confuser {

  static void A(short i) { System.out.println("Foo"); }
  static void A(int i) { System.out.println("Bar"); }

  public static void main (String[] args) {
    short i=0;
    A(i);
    A(i+i);
    A(i+=i);
  }
}
```

[Java] Surcharge (pond des râles)

Le développeur peut parfois mettre son propre grain de sel

Source (java/Confuser.java)

```
class Confuser {

  static void A(short i) { System.out.println("Foo"); }
  static void A(int i) { System.out.println("Bar"); }

  public static void main (String[] args) {
    short i=0;
    A(i);
    A(i+i);
    A(i+=i);
  }
}
```

Le programme affiche `Foo`, `Bar`, `Foo` ; dans la « vraie » vie ajoutez de l'héritage et des `Integer`

[Java] Utf ? Wtf !

Certains compilateurs sont compatibles avec l'encodage Utf-8

Source (java/Preprocess.java)

```
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 !

Certains compilateurs sont compatibles avec l'encodage Utf-8

Source (java/Preprocess.java)

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

`Bad thing happens` : le code source est donc *a priori* préprocessé préalablement à la compilation

[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

[Java] Questions d'exécution capitales

Java est un langage qui se compile en *bytecode* vérifié et interprété par une Jvm ; cela peut susciter quelques questions :

- Peut-on écrire en *bytecode* plus de choses qu'en Java ? Les vérifications sont-elles bien pensées au niveau *bytecode* ?
- Peut-on empêcher l'exécution d'un *bytecode* en restreignant les droits au niveau du système de fichiers (`chmod a-x`) ?
- Peut-on empêcher l'exécution d'un *bytecode* présent en mémoire en marquant sa page comme non exécutable ?
- La Jvm est-elle compatible avec les mécanismes de prévention d'exécution de certaines pages mémoire ?
- Quelle relation entre privilèges du *bytecode* et de la Jvm ?

Bref quelle sont les conséquences sur l'efficacité ou la possibilité de mettre en œuvre des mécanismes de sécurité système ?

[Java] Le facteur humain (sonne toujours 3 fois)

Cf. `thedailywtf.com/Articles/Java-Destruction.aspx`

Source (java/Destruction.java)

```
public class Destruction {
    public static void delete (Object object) {
        object = null;
    }
}
```

Plus de 160 commentaires en réponse, dont

- *Obviously the problem is that he forgot to call* `System.gc()`
- *It allows the object to be garbage collected. . . It is still a [problem] because a one liner should be done by the caller*
- *You know nulling objects to "help" the garbage collector is usually considered bad practice, right ?*
- *Maybe the coder was used to VB, where parameters are sent ByRef by default ?*

Plan

[JavaScript] Certains sont plus égaux que d'autres

JavaScript offre 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 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`

[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

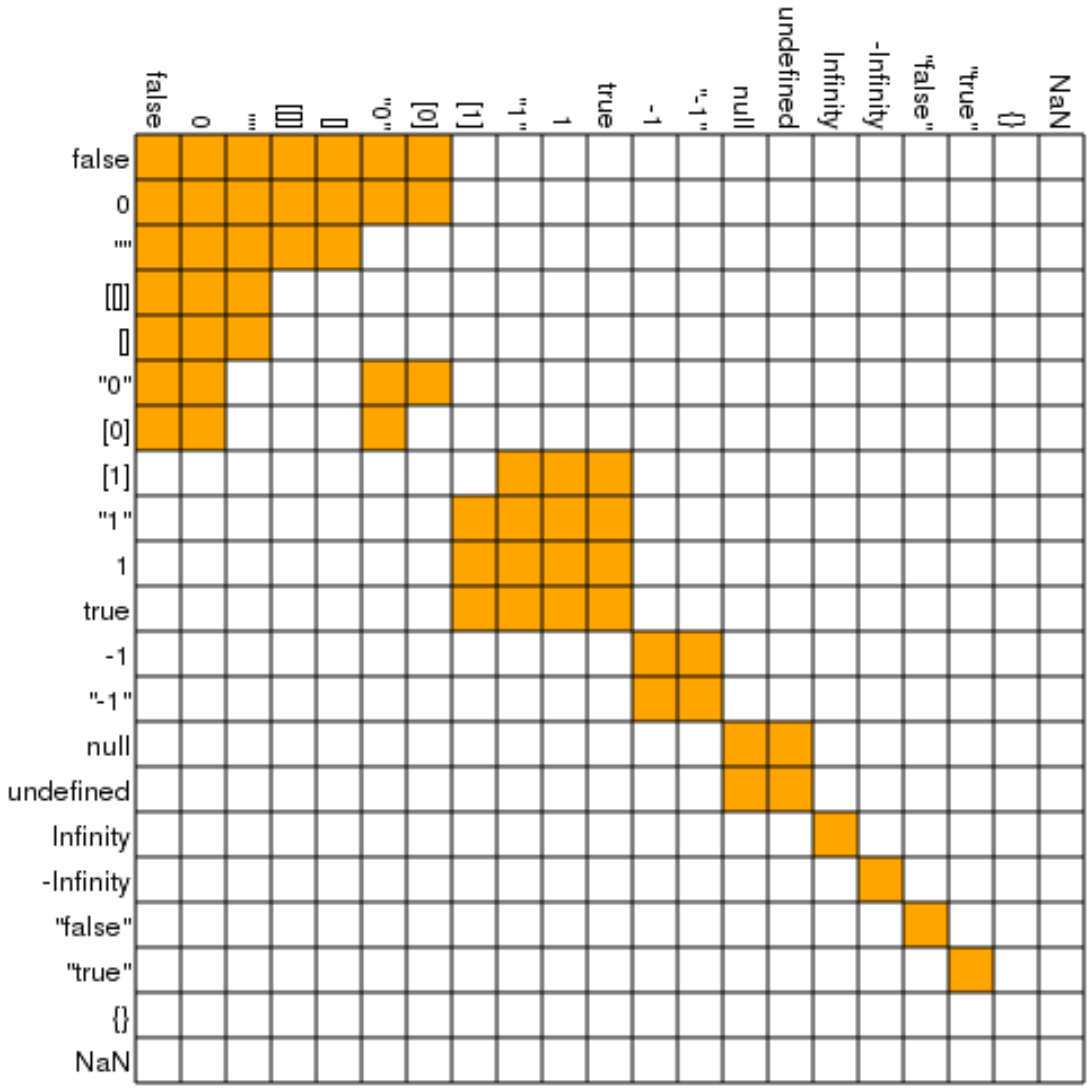
Autre exemple avec l'opérateur `+`, qui peut être l'addition d'entiers ou la concaténation de chaînes, mais qui est dans les deux cas associatif

Source (js/cast3.js)

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

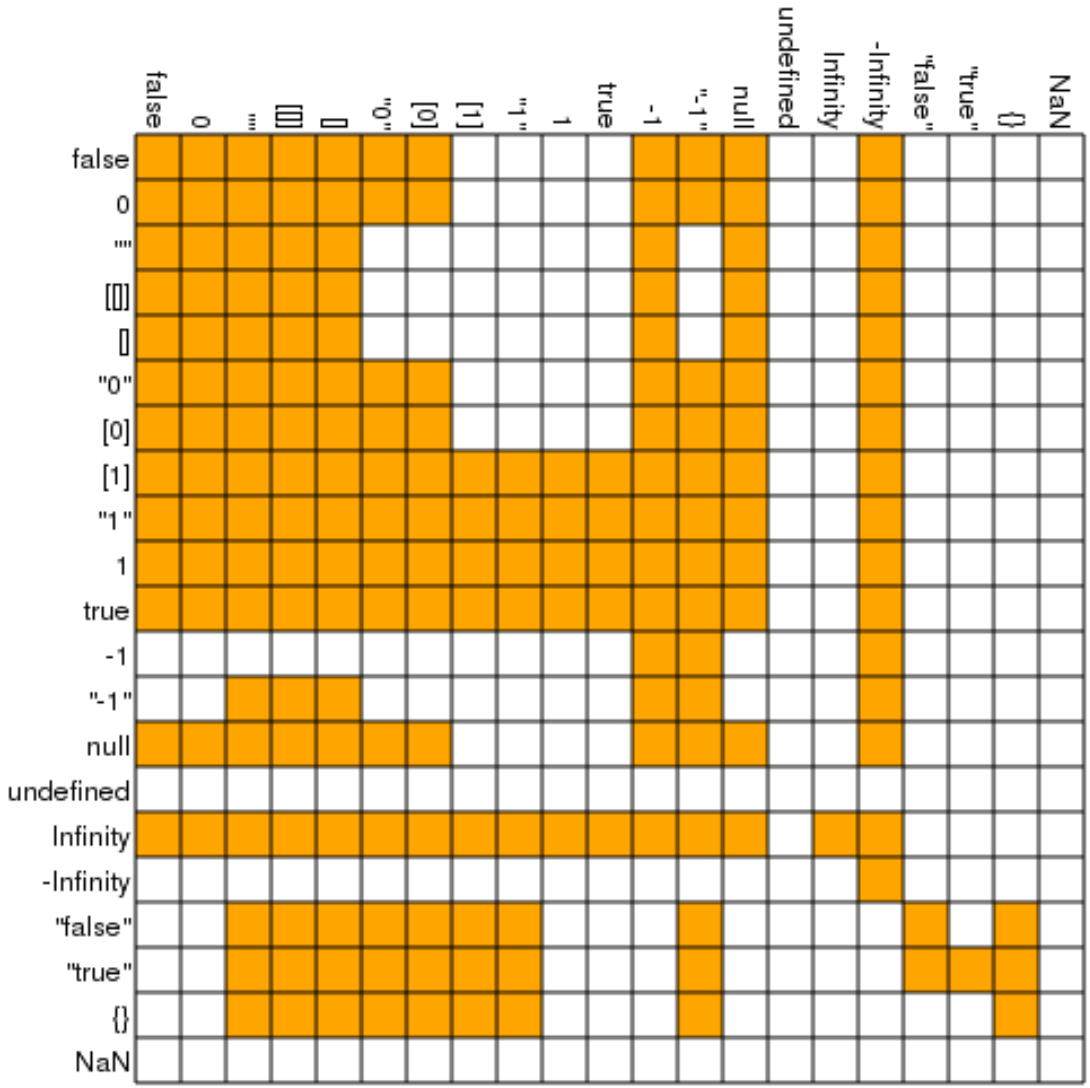
`3Foo`, `Foo12` et `Foo3`

[JavaScript] *Enter the Matrix* 1/4



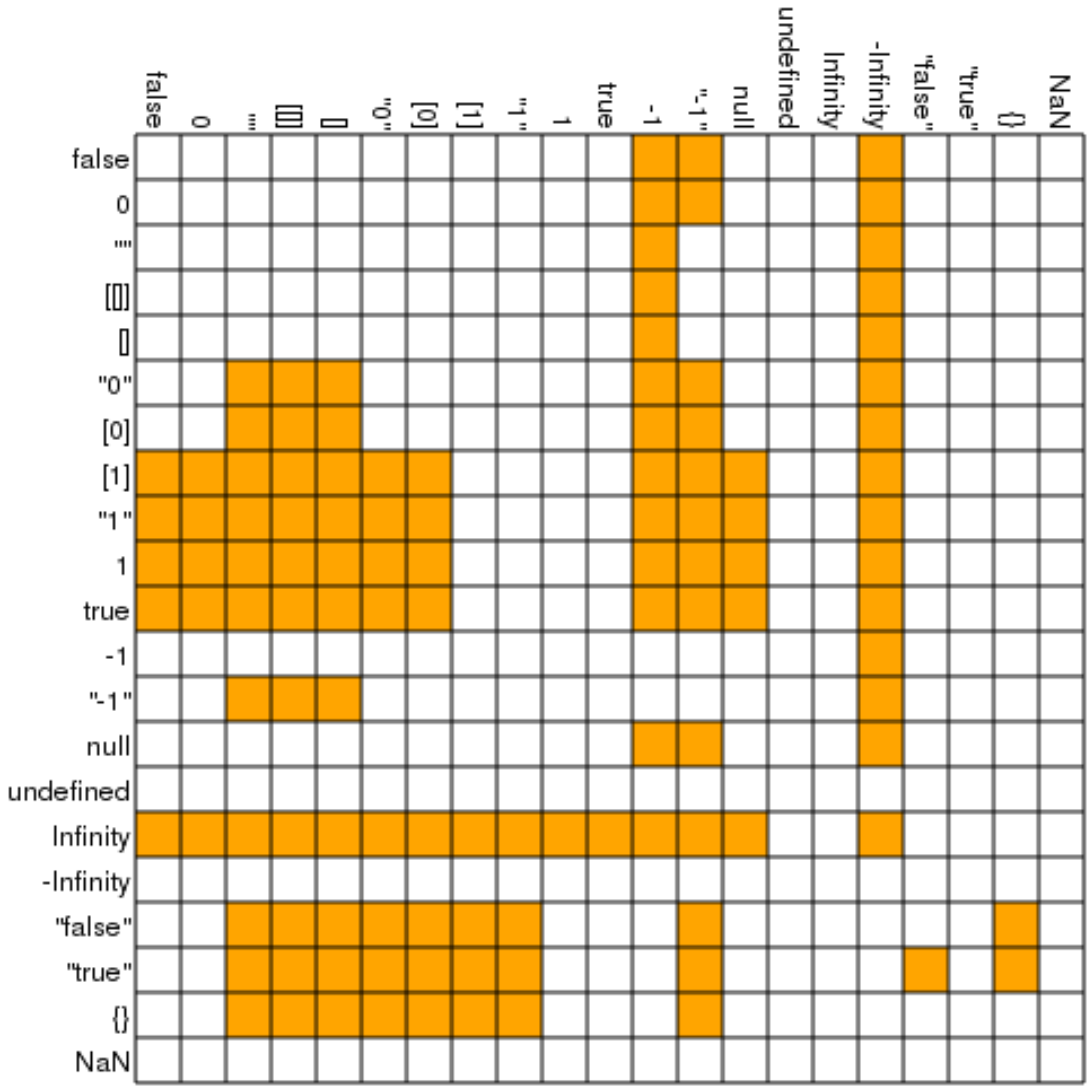
Égal ==

[JavaScript] *Enter the Matrix* 2/4



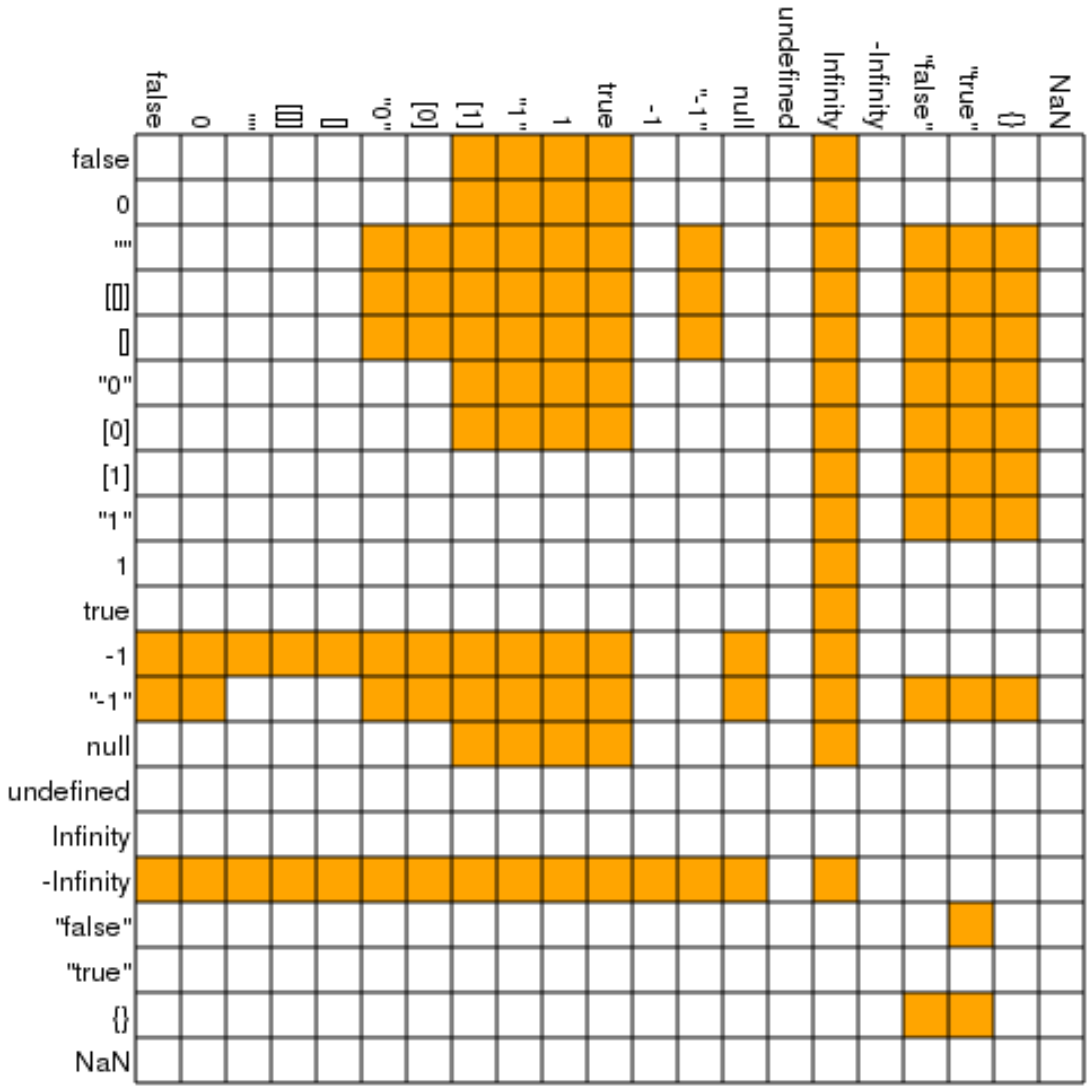
Plus petit ou égal <=

[JavaScript] *Enter the Matrix* 3/4



Plus petit <

[JavaScript] *Enter the Matrix* 4/4



Plus grand >

[PHP] Icônocast 1/2

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"); }
```

[PHP] Icônocast 1/2

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)

[PHP] Icônocast 1/2

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)

[Php] Icônocast 1/2

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`

[PHP] Icônocast 2/2

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");
```

[PHP] Icônocast 2/2

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

Plan

[OCaml] < mais costaud 1/3

OCaml offre différents mécanismes d'encapsulation [4]

Source (ocaml/hsm.ml)

```
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;;
```

C'est un boîte fermée ; la valeur `id` est visible et celle de `key` masquée

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

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

4. Ici les modules, sachant que les objets d'OCaml sont plus « fragiles »

[OCaml] < mais costaud 2/3

Mais cette encapsulation peut être contournée

Source (ocaml/hsmoracle.ml)

```
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;;
```

À l'exécution, la valeur de `key` est retournée ; impossible d'ouvrir la boîte, mais on peut la comparer à d'autres sur une balance

[OCaml] < mais costaud 3/3

Pas convaincu ? Remettons en cause le typage, alors. . .

Source (ocaml/hsm5.ml)

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

module C : Crypto  =
struct
  let id=42
  let secretchar='k'
end;;

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);;
```

L'oracle retourne `107`, le code ASCII du caractère `'k'`

[OCaml] *Mutatis mutandis* 1/2

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;;
```

[OCaml] *Mutatis mutandis* 1/2

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"`

[OCaml] *Mutatis mutandis* 2/2

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

Plan

[Php/Sql] L'exemple canon

On peut en Php faire appel à un interpréteur Sql

Source (php/injectionsql.php)

```
$dbc=mysqli_connect(HST,LOG,PWD,"School");
$cmd="SELECT * FROM Students WHERE id='".$val."'";
$dbr=mysqli_query($dbc,$cmd);
```

[Php/Sql] L'exemple canon

On peut en Php faire appel à un interpréteur Sql

Source (php/injectionsql.php)

```
$dbc=mysqli_connect(HST,LOG,PWD,"School");
$cmd="SELECT * FROM Students WHERE id='".$val."'";
$dbr=mysqli_query($dbc,$cmd);
```

Bien entendu, si `$val="Bobby'; DROP TABLE Students; //"...`

Cette notion d'injection est très générique, dans le domaine des applications *web* (Xss, Csrf...) comme ailleurs

[OCaml/*Shell*] Injection létale

L'injection résulte donc de l'utilisation d'un interpréteur, en général celui d'un autre langage

Source (ocaml/syscommand.ml)

```
let printfile filename =
  Sys.command("cat "^filename);;
```

5. Les auteurs déclinent toute responsabilité en cas d'essais

[OCaml/*Shell*] Injection létale

L'injection résulte donc de l'utilisation d'un interpréteur, en général celui d'un autre langage

Source (ocaml/syscommand.ml)

```
let printfile filename =
  Sys.command("cat "^filename);;
```

`printfile "texput.log"` aura le résultat escompté

`printfile "--version ; cd / ; rm -ri ."` sans doute pas[5]

5. Les auteurs déclinent toute responsabilité en cas d'essais

[Ruby/*Shell*] Ceci n'est pas un *pipe*

En Ruby, `Kernel.open` et `File.open` permettent d'ouvrir un fichier et ont presque le même comportement. . . Le premier (celui invoqué par `open`) permet en fait de fichier de récupérer la sortie d'une commande *Shell*

Source (ruby/injectionshell.rb)

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

Sur quel critère ? Le fait que le nom de fichier commence par le caractère `|`

[*Shell*] *Star wars*

Une fois le concept d'injection bien compris, on en arrive à se poser des questions curieuses par exemple sur le *Shell*. . .

On peut utiliser `*` en paramètre dans une ligne de commande ; *a priori* c'est simple, pourtant il peut y avoir des questions assez subtiles, par exemple que se passe-t-il s'il existe un fichier nommé `"*"`, `"-o"`, `"> rights.acl"` ou encore `"; rm *"` ?

[*Shell*] *Star wars*

Une fois le concept d'injection bien compris, on en arrive à se poser des questions curieuses par exemple sur le *Shell*...

On peut utiliser `*` en paramètre dans une ligne de commande ; *a priori* c'est simple, pourtant il peut y avoir des questions assez subtiles, par exemple que se passe-t-il s'il existe un fichier nommé `"*"`, `"-o"`, `"> rights.acl"` ou encore `"; rm *"` ?

Rien...si ce n'est qu'un fichier dont le nom commence par `"-"` sera généralement interprété *par la commande appelée* comme une option

[*Shell*] *Star wars*

Une fois le concept d'injection bien compris, on en arrive à se poser des questions curieuses par exemple sur le *Shell*...

On peut utiliser `*` en paramètre dans une ligne de commande ; *a priori* c'est simple, pourtant il peut y avoir des questions assez subtiles, par exemple que se passe-t-il s'il existe un fichier nommé `"*"`, `"-o"`, `"> rights.acl"` ou encore `"; rm *"` ?

Rien...si ce n'est qu'un fichier dont le nom commence par `"-"` sera généralement interprété *par la commande appelée* comme une option

Un autre détail : si vous exécutez `cat foo*` dans un répertoire ne contenant aucun fichier de la forme `foo*`, vous savez ce que le *Shell* passe en paramètre à la commande `cat` ?

Plan

Simple affichage

```
#include <stdio.h>

int main () {
  char* s = "toto";
  printf ("%s\n", s);
  s[1] = 'i';
  printf ("%s\n", s);
  return 0;
}
```

Simple affichage

```
#include <stdio.h>

int main () {
  char* s = "toto";
  printf ("%s\n", s);
  s[1] = 'i';
  printf ("%s\n", s);
  return 0;
}
```

`-Wwrite-strings`

Plan

Retour sur goto fail

```
/* 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;
```

Retour sur goto fail

```
#include <stdio.h>

int main () {
    printf ("Bonjour\n");
    return 0;
    printf ("Au revoir\n");
    return 1;
}
```

Retour sur goto fail

```
#include <stdio.h>

int main () {
    printf ("Bonjour\n");
    return 0;
    printf ("Au revoir\n");
    return 1;
}
```

```
-Wunreachable-code
```

Retour sur goto fail

```
#include <stdio.h>

int main () {
    printf ("Bonjour\n");
    return 0;
    printf ("Au revoir\n");
    return 1;
}
```

-Wunreachable-code (mais seulement avec clang)

Plan

Encore un problème d'affichage

```
#include <stdio.h>

void print (char* s) {
    long res=0x123456789ABCDEF0;
    long* ptr=&res;
    printf (s);
    printf ("\n%lx %p\n", res, ptr);
}

int main (int argc, char* argv[]) {
    if (argc != 2)
        return 1;
    print (argv[1]);
    return 0;
}
```

Encore un problème d'affichage

```
#include <stdio.h>

void print (char* s) {
    long res=0x123456789ABCDEF0;
    long* ptr=&res;
    printf (s);
    printf ("\n%lx %p\n", res, ptr);
}

int main (int argc, char* argv[]) {
    if (argc != 2)
        return 1;
    print (argv[1]);
    return 0;
}
```

`-Wformat=2` (ou `-Wformat -Wformat-security`)

Conclusion 1/3

- Présentation orientée vers les langages, leur syntaxe et leur sémantique
- Pièges nombreux et variés
  - un danger commun : surprendre le développeur et contredire l'intuition
- Attention : l'écriture du code n'est qu'une étape dans le cycle de développement

Il est donc essentiel de bien connaître le(s) langage(s) qu'on utilise

Conclusion 2/3

Pour bien comprendre ces exemples et leur impact sur un programme réel, des fondamentaux en informatique sont nécessaires

- sémantique des langages
- théorie de la compilation
- architecture des ordinateurs
- principes des systèmes d'exploitation
- ...

Conclusion 3/3

Réflexions personnelles sur l'enseignement du développement

- importance des fondamentaux
- utilité de voir plusieurs langages
- besoin de transmettre au plus tôt les bons réflexes
  - pas un projet C sans `-Wall -Wextra -Werror` (...)
  - aller au-delà du test unitaire (non régression, tests négatifs)
  - faire relire son code (y compris par soi-même, 1 mois plus tard)
- faire comprendre les failles et les attaques...
- ... mais sans oublier qu'il est plus dur de construire que de casser

Questions ?

Merci pour votre attention