Mind your Language!

Langages de développement et sécurité

Olivier Levillain & Éric Jaeger

Méthodes formelles et langages pour le développement de logiciels fiables dans l'industrie, lundi 28 avril 2014

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

Le sujet des langages de programmation alternatifs semble d'intérêt aujorud'hui. Selon nous, la question est *vaste*.

Sécurité et langages

Quelques aspects intéressants d'un langage en termes de sécurité

- Faux amis ou pièges
- Constructions non spécifiées ou non définies mais utilisables
- Possibilités d'offuscation pour le développeur malicieux
- Bugs dans les outils de développement
- Comportements inappropriés des outils de développement
- Limitations des capacités d'analyse
- Surprises à l'exécution
- ...

Avertissements

Les exemples qui suivent ne sont pas forcément des *bugs*, mais des situations qui ≪ attirent l'attention ≫ des experts en sécurité

Les exemples sont écrits dans certains langages de programmation

- l'objectif n'est pas de dénigrer ou d'encenser tel ou tel langage
- les concepts sont souvent *portables* à d'autres langages

Plan

[Java] Charge static

Le comportement de programmes objet très simples est parfois difficilement prévisible : que fait le code suivant [1] ?

Source (snippets/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");
  }
}
```

1. Indice : `Mathf.pi=3.1415`

[Java] Charge static

Le comportement de programmes objet très simples est parfois difficilement prévisible : que fait le code suivant [1] ?

Source (snippets/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 StaticInit
Bad things happen!
```

1. Indice : `Mathf.pi=3.1415`

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

Source (snippets/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

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

Source (snippets/ocaml/mutable.ml)

```
let check c =
  if c then "Tout va bien" else "Tout va mal!";;

let f=check false in
  f.[8]<-'b'; f.[9]<-'i'; f.[10]<-'e'; f.[11]<-'n';;

check true;;
check false;;
```

[OCaml] Mutatis mutandis

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

Source (snippets/ocaml/mutable.ml)

```
let check c =
  if c then "Tout va bien" else "Tout va mal!";;

let f=check false in
  f.[8]<-'b'; f.[9]<-'i'; f.[10]<-'e'; f.[11]<-'n';;

check true;;
check false;;
```

Les deux applications de `check` renvoient `"Tout va bien"`

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 (snippets/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

[C] Sauf exception...

Au préalable, deux petits rappels

Source (snippets/c/strategy1.c)

```
#define abs(X) (X)>=0?(X):(-X)
int abs(int x) { return x>=0?x:-x; }

#define first(x,y) x
int first(int x,int y) { return x; }
```

[C] Sauf exception...

Au préalable, deux petits rappels

Source (snippets/c/strategy1.c)

```
#define abs(X) (X)>=0?(X):(-X)
int abs(int x) { return x>=0?x:-x; }

#define first(x,y) x
int first(int x,int y) { return x; }
```

Les deux versions de `abs` n'ont pas le même comportement par exemple pour `abs(x++)`

[C] Sauf exception...

Au préalable, deux petits rappels

Source (snippets/c/strategy1.c)

```
#define abs(X) (X)>=0?(X):(-X)
int abs(int x) { return x>=0?x:-x; }

#define first(x,y) x
int first(int x,int y) { return x; }
```

Les deux versions de `abs` n'ont pas le même comportement par exemple pour `abs(x++)`

Les deux versions de `first` n'ont pas le même comportement par exemple pour `first(x,1/x)` avec `x` valant `0` – les exceptions sont également des effets de bord

[C] Sauf exception...

Ces subtilités étant clarifiées, que font les deux codes suivants ?

Source (snippets/c/strategy3.c)

```
#include <stdio.h>
int zero(int x) { return 0; }
int main(void) { int x=0; x=zero(1/x); return 0; }
```

Source (snippets/c/strategy3b.c)

```
#include <stdio.h>
int zero(int x) { return 0; }
int main(void) { int x=0; return zero(1/x); }
```

[C] Sauf exception...

Ces subtilités étant clarifiées, que font les deux codes suivants ?

Source (snippets/c/strategy3.c)

```
#include <stdio.h>
int zero(int x) { return 0; }
int main(void) { int x=0; x=zero(1/x); return 0; }
```

Source (snippets/c/strategy3b.c)

```
#include <stdio.h>
int zero(int x) { return 0; }
int main(void) { int x=0; return zero(1/x); }
```

Tout dépend du niveau d'optimisation :

- `-O0` : `Floating point exception (core dumped)`
- `-O1` : le premier programme termine normalement
- `-O2` : les deux programmes terminent normalement

[JavaScript] Certains sont plus égaux que d'autres

JavaScript offre également tout le confort moderne...

Source (snippets/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 (snippets/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 ?

Source (snippets/js/cast2.js)

```
if ('0'==0) print("'0'==0");
else print("'0'<>0");
if (0=='0.0') print("0=='0.0'");
else print("0<>'0.0'");
if ('0'=='0.0') print("'0'=='0.0'");
else print("'0'<>'0.0'");
```

[JavaScript] Reconversion

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

Source (snippets/js/cast2.js)

```
if ('0'==0) print("'0'==0");
else print("'0'<>0");
if (0=='0.0') print("0=='0.0'");
else print("0<>'0.0'");
if ('0'=='0.0') print("'0'=='0.0'");
else print("'0'<>'0.0'");
```

`'0'==0`, `0=='0.0'` et `'0'<>'0.0'`

[JavaScript] Reconversion

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

Source (snippets/js/cast2.js)

```
if ('0'==0) print("'0'==0");
else print("'0'<>0");
if (0=='0.0') print("0=='0.0'");
else print("0<>'0.0'");
if ('0'=='0.0') print("'0'=='0.0'");
else print("'0'<>'0.0'");
```

`'0'==0`, `0=='0.0'` et `'0'<>'0.0'`

Source (snippets/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 ?

Source (snippets/js/cast2.js)

```
if ('0'==0) print("'0'==0");
else print("'0'<>0");
if (0=='0.0') print("0=='0.0'");
else print("0<>'0.0'");
if ('0'=='0.0') print("'0'=='0.0'");
else print("'0'<>'0.0'");
```

`'0'==0`, `0=='0.0'` et `'0'<>'0.0'`

Source (snippets/js/cast3.js)

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

`3X`, `X12` et `X3`

[JavaScript] Offusquez-vous !

Surcharges et *casts* permettent au compilateur de tripatouiller le code jusqu'à lui trouver *un*[2] sens

Source (snippets/js/weirdeval.js)

```
{} + {}
[] + {}
{} + []
({} + {})
```

Toutes ces expressions ont un sens différent de celui des autres (même pour la première et la quatrième)

2. Article indéterminé

[SQL] Transgresser un ordre

Certains types sont peut être trop complexes pour que la transitivité puisse rester vraie

Source (snippets/sql/partial.sql)

```
SELECT CONCAT(IF(@X<=@Y,'X<=Y','X>Y'),
              ' and ',
              IF(@X>=@Y,'X>=Y','X<Y')) AS Test;
```

Avec `SET @X=1; SET @Y=2;` on obtient `X<=Y and X<Y`

[SQL] Transgresser un ordre

Certains types sont peut être trop complexes pour que la transitivité puisse rester vraie

Source (snippets/sql/partial.sql)

```
SELECT CONCAT(IF(@X<=@Y,'X<=Y','X>Y'),
              ' and ',
              IF(@X>=@Y,'X>=Y','X<Y')) AS Test;
```

Avec `SET @X=1; SET @Y=2;` on obtient `X<=Y and X<Y`

Avec `SET @X=NULL` on obtient `X>Y and X<Y` (le même type d'observations peut être fait avec les flottants et `NaN`)

[OCaml] Certains sont plus égaux que d'autres

En OCaml, `x` et `y` étant des valeurs, combien de valeurs peut prendre l'expression booléenne `x=y` ?

[OCaml] Certains sont plus égaux que d'autres

En OCaml, `x` et `y` étant des valeurs, combien de valeurs peut prendre l'expression booléenne `x=y` ?

Source (snippets/ocaml/bool4.ml)

```
type tree= Leaf | Node of tree*tree;;

let zero = Leaf;;
let one = Node(Leaf,Leaf);;
let rec many = Node(Leaf,many);;
let rec toomuch = Node(toomuch,toomuch);;
```

`zero=zero` donne `true`, `zero=one` donne `false`, `many=many` boucle et `toomuch=toomuch` provoque une erreur

[Shell] Interdiction d'interdire

Pas convaincu par la remarque précédente ? Et pourtant...

Source (snippets/shell/login.sh)

```
#!/bin/bash
PIN=1234
echo -n "Veuillez saisir le code PIN (4 chiffres): "
read -s PIN_SAISI; echo

if [ "$PIN" -ne "$PIN_SAISI" ]; then
    echo "Code PIN invalide."; exit 1
else
    echo "Authentification OK"; exit 0
fi
```

[Shell] Interdiction d'interdire

Pas convaincu par la remarque précédente ? Et pourtant...

Source (snippets/shell/login.sh)

```
#!/bin/bash
PIN=1234
echo -n "Veuillez saisir le code PIN (4 chiffres): "
read -s PIN_SAISI; echo

if [ "$PIN" -ne "$PIN_SAISI" ]; then
    echo "Code PIN invalide."; exit 1
else
    echo "Authentification 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] Un `goto fail` peut en cacher un autre

En mars dernier, juste après la faille `goto fail` Apple, une faille aux conséquences semblables a été découverte dans GnuTLS :

> *[This bug] 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] Un `goto fail` peut en cacher un autre

En mars dernier, juste après la faille `goto fail` Apple, une faille aux conséquences semblables a été découverte dans GnuTLS :

> *[This bug] 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.*

Un problème similaire dans OpenSSL : CVE-**2008**-5077

[PHP/SQL] L'exemple canon

On peut en PHP faire appel à un interpréteur SQL

Source (snippets/php/injectionsql.php)

```
<?php
function readline($val)
{ $dbc=mysqli_connect(HST,LOG,PWD,"School") or die("err1")
    ;
  $cmd="SELECT * FROM Students WHERE id='".$val."'";
  $dbr=mysqli_query($dbc,$cmd) or die("err2");
  $res=array();
  while ($row=mysqli_fetch_row($dbr)) $res[]=$row;
  mysqli_free_result($dbr); mysqli_close($dbc);
  return $res;
}
?>
```

[PHP/SQL] L'exemple canon

On peut en PHP faire appel à un interpréteur SQL

Source (snippets/php/injectionsql.php)

```
<?php
function readline($val)
{ $dbc=mysqli_connect(HST,LOG,PWD,"School") or die("err1")
    ;
  $cmd="SELECT * FROM Students WHERE id='".$val."'";
  $dbr=mysqli_query($dbc,$cmd) or die("err2");
  $res=array();
  while ($row=mysqli_fetch_row($dbr)) $res[]=$row;
  mysqli_free_result($dbr); mysqli_close($dbc);
  return $res;
}
?>
```

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

[OCaml/Shell] Injection létale

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

Source (snippets/ocaml/syscommand.ml)

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

[OCaml/Shell] Injection létale

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

Source (snippets/ocaml/syscommand.ml)

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

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

`printfile "--version ; cd / ; rm -fr ."` sans doute pas – ne testez pas !

[Php] Moins d'évaluation pour plus de sécurité

Certains langages interprétés intègrent un évaluateur qui permet de construire dynamiquement un programme à partir d'une valeur

Source (snippets/php/injectioneval.php)

```
<?php

$cmd1="echo 'Hello world<br />';";
$cmd2="die('eval killed me');";
$cmd3="echo 'Good bye<br />';";

eval($cmd1.$cmd2.$cmd3);

?>
```

[PHP] Moins d'évaluation pour plus de sécurité

Certains langages interprétés intègrent un évaluateur qui permet de construire dynamiquement un programme à partir d'une valeur

Source (snippets/php/injectioneval.php)

```
<?php

$cmd1="echo 'Hello world<br />';";
$cmd2="die('eval killed me');";
$cmd3="echo 'Good bye<br />';";

eval($cmd1.$cmd2.$cmd3);

?>
```

```
Hello world

eval killed me
```

C'est bien sûr dangereux, mais aussi quasiment impossible à analyser

[Python] Localité en fête

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

Source (snippets/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 liste en compréhension

Source (snippets/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).

[OCaml] Sans commentaires

Il est possible d'ouvrir une chaîne de caractères dans un commentaire OCaml, ce qui peut induire en erreur un relecteur, surtout si la coloration syntaxique n'est pas conforme

Source (snippets/ocaml/comments.ml)

```
(* blah blah " blah blah *)
let x=true;;

(* PREVIOUS VERSION ----------------------
(* blah blah " blah blah *)
let x=false;;
(* blah blah " blah blah *)
----------------------------------------*)
(* blah blah " blah blah *)
```

Ce qui semble être en commentaire ne l'est pas, et *vice versa* ; à noter que l'exemple donné ici est volontairement non offusqué

[C] Toujours pas de commentaires

Les commentaires semblent délicats à traiter dans d'autres langages

Source (snippets/c/comments.c)

```
#include <stdio.h>

int foo() {
  int a=4; int b=2;
  return a //*
           //*/ b
  ;
}

int main(void) {
  printf("%d\n",foo()); return 0;
}
```

Ce code, compilé et exécuté, affiche `4` ; mais si on compile en mode C89 (option `-std=c89` de GCC) on obtient `2`

[Java] Clone Wars

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

```
protected Object clone()
```

...

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.*

...

[Java] Clone Wars

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

```
protected Object clone()
```

...

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.*

...

Dans un autre registre, la spécification des opérations de sérialisation (`writeObject` et `readObject`) est aussi assez. . . disons 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

C'est fait ?

[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'est fait ?

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*

[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 ont-elles été pensées au niveau source ou *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 ?
- Réciproquement la Jvm est-elle compatible avec les mécanismes de prévention d'exécution de certaines pages mémoire ?
- Quelle relation entre privilèges du *bytecode* et ceux de la Jvm ?

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

[OCaml] Mémoire collective et amnésie

OCaml met en œuvre un Gc qui gère la mémoire ; supposons que nous voulions implémenter très proprement une bibliothèque cryptographique manipulant des clés secrètes et/ou privées

- Comment interdire les copies (compactage ou *swap*) ?
- Comment minimiser la durée de présence d'une clé en mémoire ?
- Comment effacer une clé par surcharge ?

Au passage, notons qu'un mécanisme non fonctionnel [3] tel que la surcharge peut aussi être victime d'optimisations de compilation, de mécanismes de cache, de technologies telles que celle des mémoires *flash*, *etc.*

3. C'est à dire sans effet visible sur les résultats de l'exécution

Plan

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
  - ...

A 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 [4]
- *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

4. Voire formalisée...

A 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 [5]
  - Modèle mémoire reflétant l'encapsulation
  - Surveiller le flot d'exécution même en présence de fautes
- Disposer d'outils maîtrisés voire de confiance

5. Cela peut aussi concerner les architectures. . .

Remerciements

Les exemples de cette présentation ont étés 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 ;-)

Outils et langages utilisés

- C : `gcc (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3`
- Erlang : `Erlang R14B04 (erts-5.8.5)`
- Java : `Eclipse Java Compiler 0.972_R35x, 3.5.1 release`
- JavaScript : `Mozilla Firefox 16.0.2 for Ubuntu canonical - 1.0`
- SQL : `mysql Ver 14.14 Distrib 5.5.28, for debian-linux-gnu (i686)`
- OCaml : `The Objective Caml compiler, version 3.12.1`
- Php : `Server version: Apache/2.2.22 (Ubuntu)`
- Python 3
- *Shell* : `GNU bash, version 4.2.24(1)-release (i686-pc-linux-gnu)`