Langages de développement et sécurité

Mind your language!

Eric Jaeger & Olivier Levillain

Séminaire Lrde, Epita – 29 mai 2013

ANSSI

Plan

Motivation historique

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

Une question intéressante, et qui peut être généralisée...

- Certains langages sont-ils plus appropriés que d'autres ? Selon quels critères ? Que serait un langage dédié à la sécurité ?
- Faut-il interdire, déconseiller, recommander ou imposer certaines constructions ou configurations ?
- Qu'attendre du compilateur, des outils, du *runtime* ?
- Comment adapter l'évaluation indépendante ?

Il semble n'y avoir que peu de travaux sur le thème de la sécurité alors que de nombreuses choses existent en ce qui concerne la sûreté

Une citation

Extrait du discours de *C.A.R.* Hoare à l'occasion de la remise de son *Turing Award* en 1980

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.

Objectifs de la présentation

Inciter à la réflexion et permettre une prise de conscience dans le domaine de la sécurité – sans imposer ou interdire

Mettre en évidence la question du choix du langage, rarement traitée de manière satisfaisante : après tout, les caractéristiques du langage utilisé pourraient faire partie de la solution. . . et non pas du problème

Proposer des pistes sur ce que serait un langage ou des outils dédiés à la sécurité, ou encore les spécificités de la formation des développeurs de systèmes critiques, *etc.*

Plan

Avertissement préalable

Les extraits de code suivants illustrent certains problèmes de sécurité

- Constructions dangereuses dont l'intérêt semble limité
- Faux amis ou vrais pièges pour un développeur peu attentif
- Perte de lisibilité ou traçabilité pour un évaluateur indépendant
- ...

La remarque peut s'appliquer à d'autres langages que celui utilisé

Ce qui est mis en évidence n'est pas toujours aberrant, mais doit inciter à se poser des questions quand des objectifs de sécurité sont applicables

Une dernière chose : ne croyez pas que "personne ne fait ça !", vous pourriez être surpris de ce qui traîne dans la nature

Plan

[Java] Charge static 1/2

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

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");
  }
}
```

1. Indice : `Mathf.pi=3.1415`

[Java] Charge static 1/2

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

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");
  }
}
```

Voici ce que donne l'exécution :

```
demo$ java StaticInit
Bad things happen!
```

1. Indice : `Mathf.pi=3.1415`

[Java] Charge static 2/2

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

Source (java/Mathf.java)

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

```
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); }
}
```

Il semble possible que sur certaines implémentations de la Jvm une simple déclaration sans initialisation (`Mathf dummy;`) puisse avoir des effets similaires

Nous retiendrons surtout que l'exercice de relecture d'un code Java, même élémentaire, peut se révéler assez complexe et contre-intuitif !

[Java] Serial killer 1/2

Chat échaudé craint l'eau froide, évitons 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 1/2

Chat échaudé craint l'eau froide, évitons 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 voici ce qui est affiché : `Bad things happen!`

[JAVA] Serial killer 2/2

Source (java/Deserialf.java)

```
import java.io.*;

class Deserialk implements Serializable {
  static final long serialVersionUID=0;
  static { System.out.println("Bad things happen!");
           System.exit(0); }
}

class Deserialf {
  public static void main(String[] args)
    throws FileNotFoundException, IOException {
    FileOutputStream fos=new FileOutputStream("friend");
    ObjectOutputStream oos=new ObjectOutputStream(fos);
    oos.writeObject(new Deserialk());
  }
}
```

Pour créer le fichier `friend`*, il faut désactiver le code* `static` *en le commentant avant compilation et exécution ; puis il faut décommenter et recompiler*

Plan

[JAVA] Encore une *objection*

Les langages objet offrent diverses formes d'encapsulation, mais ces mécanismes relèvent de l'ingénierie logicielle plus que de la 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); }
  }
}
```

Ce programme affiche `Secret.x=42` ; on peut interdire l'introspection dans la politique de sécurité JAVA, mais c'est assez complexe

[OCaml] < mais costaud 1/3

OCaml offre différents mécanismes d'encapsulation [2]

Source (ocaml/hsm-old.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(); let s=Random.int 8192 in
           Printf.printf "< id=%d, key=%d >\n" id s; s)
end;;
```

A l'exécution, on obtient `< id=2570, secret=6151 >`

La valeur `id` est visible, alors que `key` est masquée

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

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

2. Ici les modules, sachant que les objets d'OCaml sont plus "fragiles"

[OCaml] < mais costaud 2/3

Cette encapsulation peut être contournée (sans même utiliser `Obj`)

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 retournée est bien `- : int = 6151` ; les modules étant des valeurs en OCaml, on peut les comparer avec l'opérateur `<` qui bien que polymorphe permet une analyse structurelle

[OCaml] < mais costaud 3/3

Mais en fait à quoi bon s'embêter avec les champs ou les types ?

Source (ocaml/hsm5-old.ml)

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

module C : Crypto  =
struct
  let id=Random.self_init(); Random.int 8192
  let key='k'; (* ASCII code 107 *)
end;;

let rec oracle o1 o2 =
  let o = (o1 + o2)/2 in
  let module O = struct let id=C.id let notachar=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);;
```

En clair l'oracle compare entiers et caractères et renvoie 107 !

[JAVA] Nombrilisme 1/3

Il est possible en JAVA de définir une classe interne qui a accès aux champs privés de la classe englobante

Source (java/Innerclass2.java)

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

  static public class Innerinner {
    public static void print()
    { System.out.println(Innerclass2.secret); }
  }

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

L'affichage donne 42

[Java] Nombrilisme 2/3

L'inverse est également vrai, la classe englobante peut accéder aux champs privés de la classe interne

Source (java/Innerclass1.java)

```
public class Innerclass {
  static public class Innerinner {
    private static int secret=42;
  }

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

L'affichage donne 42

[Java] Nombrilisme 3/3

Ce qui est intéressant, c'est que la notion de classe interne n'existe pas en *bytecode*, et que `Innerinner` a donc été promue au niveau global. . . Qu'advient-il alors du `private` du champ `secret` ?

Source (java/Innerexploit.java)

```
public class Innerexploit {
  public static void main (String[] args) {
    System.out.println(Innerclass.Innerinner.secret);
  }
}
```

`Innerclass.Innerinnet.secret` *est rendu* `public` *avant de compiler* `Innerexploit`, *puis remis* `private` *avant de recompiler* `Innerclass`

L'exécution de `Innerexploit` donne bien `42`

Plan

[C] Questions stratégiques

On sait que le C permet d'écrire des choses difficiles à comprendre...

Source (c/effect-old.c)

```
#include <stdio.h>
int id(int x) { printf("%d ",x); return x; }
int main(void) {
  { int c=0; printf("++c,++c -> %d,%d\n",++c,++c); }
  { int c=0; printf("c++,c++ -> %d,%d\n",c++,c++); }
  { int c=0; printf("id(id(++c)) : "); id(id(++c)); }
  { int c=0; printf("\nid(id(c++)) : "); id(id(c++)); }
  return 0;
}
```

[C] Questions stratégiques

On sait que le C permet d'écrire des choses difficiles à comprendre...

Source (c/effect-old.c)

```
#include <stdio.h>
int id(int x) { printf("%d ",x); return x; }
int main(void) {
  { int c=0; printf("++c,++c -> %d,%d\n",++c,++c); }
  { int c=0; printf("c++,c++ -> %d,%d\n",c++,c++); }
  { int c=0; printf("id(id(++c)) : "); id(id(++c)); }
  { int c=0; printf("\nid(id(c++)) : "); id(id(c++)); }
  return 0;
}
```

La surprise ici c'est que toute tentative d'explication sera très longue !

```
++c,++c -> 2,2
c++,c++ -> 1,0
id(id(++c)): 1 1
id(id(c++)): 0 0
```

[OCaml] *Mutatis mutandis* 1/3

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

Source (ocaml/mutable-old.ml)

```
let alert test =
  if test then "Tout va bien" else "Tout va mal!";;

alert true;;
alert false;;

(alert false).[8]<-'b'; (alert false).[9]<-'i';
(alert false).[10]<-'e'; (alert false).[11]<-'n';;

alert false;;
```

[OCaml] *Mutatis mutandis* 1/3

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

Source (ocaml/mutable-old.ml)

```
let alert test =
  if test then "Tout va bien" else "Tout va mal!";;

alert true;;
alert false;;

(alert false).[8]<-'b'; (alert false).[9]<-'i';
(alert false).[10]<-'e'; (alert false).[11]<-'n';;

alert false;;
```

Voici ce qui s'affiche à l'exécution :

```
- : string = "Tout va bien"
- : string = "Tout va mal!"
- : string = "Tout va bien"
```

[OCaml] *Mutatis mutandis* 2/3

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-old.ml)

```
(string_of_bool true).[0]<-'f';
(string_of_bool true).[1]<-'a';
(string_of_bool true).[3]<-'x';
Printf.printf "1=1 s'evalue a %b\n" (1=1);;
```

[OCaml] *Mutatis mutandis* 2/3

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-old.ml)

```
(string_of_bool true).[0]<-'f';
(string_of_bool true).[1]<-'a';
(string_of_bool true).[3]<-'x';
Printf.printf "1=1 s'evalue a %b\n" (1=1);;
```

Bien entendu, à l'exécution nous obtenons `1=1 s'evalue a faux`
Cela ne marche pas avec `string_of_int`, mais d'autres fonctions intéressantes sont concernées, par exemple `Char.escaped`

[OCaml] *Mutatis mutandis* 3/3

Cela s'applique aussi aux exceptions, et certains *patterns* de développement usuels [3] deviennent dès lors dangereux

Source (ocaml/mutableexc.ml)

```
let alert test =
  if test then failwith "minor" else failwith "major";;

let reaction test =
  try ignore (alert test)
  with Failure "minor" -> ()
     | Failure "major" -> failwith "major";;

try alert false with Failure x -> (x.[1]<-'i'; x.[2]<-'n');;

reaction false;;
```

3. Cf. par exemple `char_of_int` dans la bibliothèque standard

[OCaml] *Mutatis mutandis* 3/3

Cela s'applique aussi aux exceptions, et certains *patterns* de développement usuels [3] deviennent dès lors dangereux

Source (ocaml/mutableexc.ml)

```
let alert test =
  if test then failwith "minor" else failwith "major";;

let reaction test =
  try ignore (alert test)
  with Failure "minor" -> ()
     | Failure "major" -> failwith "major";;

try alert false with Failure x -> (x.[1]<-'i'; x.[2]<-'n');;

reaction false;;
```

`reaction false` ne génère plus d'exception, le flot a été détourné

3. Cf. par exemple `char_of_int` dans la bibliothèque standard

[C] Opérations stratégiques 1/3

Le théoricien le sait : en présence d'effets de bord, la stratégie devient significative, et ce qui semble équivalent ne l'est pas forcément

Source (c/strategy-old.c)

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

int main(void) {
  int x=-2; printf("x=%d, ",x);
  printf("abs(x++)=%d (function), ",abs(x++));
  printf("x=%d\n",x);
  x=-2; printf("x=%d, ",x);
  printf("abs(x++)=%d (macro), ",_abs(x++));
  printf("x=%d\n",x);
}
```

[C] Opérations stratégiques 1/3

Le théoricien le sait : en présence d'effets de bord, la stratégie devient significative, et ce qui semble équivalent ne l'est pas forcément

Source (c/strategy-old.c)

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

int main(void) {
  int x=-2; printf("x=%d, ",x);
  printf("abs(x++)=%d (function), ",abs(x++));
  printf("x=%d\n",x);
  x=-2; printf("x=%d, ",x);
  printf("abs(x++)=%d (macro), ",_abs(x++));
  printf("x=%d\n",x);
}
```

```
x=-2, abs(x++)=2 (function), x=-1
x=-2, abs(x++)=1 (macro), x=0
```

[C] Opérations stratégiques 2/3

Il faut aussi noter que la notion d'effet de bord est très générique

Source (c/strategy2.c)

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

int main(void) {
  int x=0;
  if (_fst(x,1/x)==x) printf("Hello world\n");
  if (fst(x,1/x)==x) printf("Hello world\n");
  return 0;
}
```

[C] Opérations stratégiques 2/3

Il faut aussi noter que la notion d'effet de bord est très générique

Source (c/strategy2.c)

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

int main(void) {
  int x=0;
  if (_fst(x,1/x)==x) printf("Hello world\n");
  if (fst(x,1/x)==x) printf("Hello world\n");
  return 0;
}
```

```
Hello world
Floating point exception (core dumped)
```

[C] Opérations stratégiques 3/3

Ces subtilités étant clarifiées, que fait le code suivant ?

Source (c/strategy3-old.c)

```
#include <stdio.h>

int zero(int x) { return 0; }

int main(void) {
  int x=0;
  if (zero(1/x)==0) printf("Hello world\n");
  return 0;
}
```

[C] Opérations stratégiques 3/3

Ces subtilités étant clarifiées, que fait le code suivant ?

Source (c/strategy3-old.c)

```
#include <stdio.h>

int zero(int x) { return 0; }

int main(void) {
  int x=0;
  if (zero(1/x)==0) printf("Hello world\n");
  return 0;
}
```

Réponse : ça dépend du niveau d'optimisation à la compilation

- Jusqu'à `-O1`, `Floating point exception (core dumped)`
- À partir de `-O2`, `Hello world`

[C] Affectations

Tant qu'on en est à parler d'effets de bord, voici le plus simple

Source (c/strategy4.c)

```
int diff(int x,int y) { return (2*x)+y; }

int main(void) {
  int x=0;
  int y=diff((x=1),(x=3));
  printf("%d\n",y);
  return 0;
}
```

Ce programme affiche 3 ; ce type de code est explicitement déconseillé mais ne déclenche aucun avertissement à la compilation

Plan

[C] La goutte d'eau

Rappelons tout d'abord le principe du *Buffer Overflow*[4]

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;
}
```

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

[C] La goutte d'eau

Rappelons tout d'abord le principe du *Buffer Overflow*[4]

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

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

[C] Faire mauvaise impression 1/2

Les manipulations sur les pointeurs sont parfois bien cachées...

Source (c/stringformat.c)

```
#include <stdio.h>

int sfa() {
 char *f="%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x";
 printf(f); printf("\n"); return 0;
}

int main(void) {
  int secret=0x40414243;
  sfa();
  return 0;
}
```

[C] Faire mauvaise impression 1/2

Les manipulations sur les pointeurs sont parfois bien cachées...

Source (c/stringformat.c)

```
#include <stdio.h>

int sfa() {
 char *f="%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x";
 printf(f); printf("\n"); return 0;
}

int main(void) {
  int secret=0x40414243;
  sfa();
  return 0;
}
```

`...8048469.40414243.8048460...` : le `printf` permet de parcourir à rebours la pile au-delà du contexte lié à l'appel de `sfa`

[C] Faire mauvaise impression 2/2

Avec la balise `%n` il est même possible de corrompre la pile

Source (c/stringformat2.c)

```
#include <stdio.h>

int main(void) {
  char *f="%x.%x.%x.%x.%x.%x.%n";
  int s=0x40414243;
  int *p=&s;
  printf(f); printf("\n");
  if (s==0x40414243) printf("Hello world\n");
  else printf("Bad things happen! s is now %08x\n",s);
  return 0;
}
```

[C] Faire mauvaise impression 2/2

Avec la balise `%n` il est même possible de corrompre la pile

Source (c/stringformat2.c)

```
#include <stdio.h>

int main(void) {
  char *f="%x.%x.%x.%x.%x.%x.%n";
  int s=0x40414243;
  int *p=&s;
  printf(f); printf("\n");
  if (s==0x40414243) printf("Hello world\n");
  else printf("Bad things happen! s is now %08x\n",s);
  return 0;
}
```

```
51b196.68dff4.51b225.2c1270.40414243.8048580.
Bad things happen! s is now 0000002d
```

Plan

[Erlang] *Let it crash*

Erlang, comme d'autres langages, aide le développeur en proposant des surcharges d'opérateurs et des conversions automatiques (*cast*) ; ainsi `1+1`, `1.0+1.0` et `1+1.0` sont des expressions valides, mais il y a de légitimes questions quant à la cohérence de la démarche

Source (erlang/factorial.erl)

```
-module(factorial).
-compile(export_all).

fact(0) -> 1;
fact(N) -> N*fact(N-1).
```

[Erlang] *Let it crash*

Erlang, comme d'autres langages, aide le développeur en proposant des surcharges d'opérateurs et des conversions automatiques (*cast*) ; ainsi `1+1`, `1.0+1.0` et `1+1.0` sont des expressions valides, mais il y a de légitimes questions quant à la cohérence de la démarche

Source (erlang/factorial.erl)

```
-module(factorial).
-compile(export_all).

fact(0) -> 1;
fact(N) -> N*fact(N-1).
```

```
demo$ erl
1> factorial:fact(4).
24
2> factorial:fact(4.0).
eheap_alloc: Cannot allocate...
```

[JavaScript] Certains sont plus égaux que d'autres

JavaScript offre également tout le confort moderne...

Source (js/unification.js)

```
document.write("0",
               0=='0'?"==":"<>",
               "'0'");

document.write(" and ");

switch (0)
{ case '0':document.write("0=='0'<br />");
  default:document.write("0<>'0'<br />");
}
```

[JavaScript] Certains sont plus égaux que d'autres

JavaScript offre également tout le confort moderne...

Source (js/unification.js)

```
document.write("0",
               0=='0'?"==":"<>",
               "'0'");

document.write(" and ");

switch (0)
{ case '0':document.write("0=='0'<br />");
  default:document.write("0<>'0'<br />");
}
```

L'affichage obtenu est `0=='0' and 0<>'0'`

[JavaScript] Reconversion

À choisir, faut-il favoriser *cast* et surcharge, ou la préservation des propriétés usuelles telles que associativité et transitivité ?

Source (js/cast.js)

```
document.write("'0'",'0'==0?"==":"<>","0 and ");
document.write("0",0=='0.0'?"==":"<>","'0.0' and ");
document.write("'0'",'0'=='0.0'?"==":"<>","'0.0'<br />");

document.write(1+2+'X'); document.write(' and ');
document.write('X'+1+2); document.write(' and ');
document.write('X'+(1+2)); document.write('<br />');
```

[JavaScript] Reconversion

À choisir, faut-il favoriser *cast* et surcharge, ou la préservation des propriétés usuelles telles que associativité et transitivité ?

Source (js/cast.js)

```
document.write("'0'",'0'==0?"==":"<>","0 and ");
document.write("0",0=='0.0'?"==":"<>","'0.0' and ");
document.write("'0'",'0'=='0.0'?"==":"<>","'0.0'<br />");

document.write(1+2+'X'); document.write(' and ');
document.write('X'+1+2); document.write(' and ');
document.write('X'+(1+2)); document.write('<br />');
```

```
'0'==0 and 0=='0.0' and '0'<>'0.0'
3X and X12 and X3
```

Les constantes aident à comprendre ; par contre, si en plus on se met à utiliser des variables, le comportement semble, au mieux, cryptique. . .

[C] Système de *cast*

Le (compilateur) C cherche également à rendre service au développeur en essayant de deviner ce qu'il veut

Source (c/cast0.c)

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

[C] Système de *cast*

Le (compilateur) C cherche également à rendre service au développeur en essayant de deviner ce qu'il veut

Source (c/cast0.c)

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

```
3/4=0.000000
```

[C] Un développeur averti en vaut deux (1/2)

Certains *cast* sont peu visibles et inévitables

Source (c/castregister.c)

```
#include <stdio.h>

int main(void) {
  unsigned char x = 128;
  unsigned char y = 2;
  unsigned char z = (x * y) / y;
  printf("(%d * %d) / %d = %d\n",x,y,y,z);
  return 0;
}
```

[C] Un développeur averti en vaut deux (1/2)

Certains *cast* sont peu visibles et inévitables

Source (c/castregister.c)

```
#include <stdio.h>

int main(void) {
  unsigned char x = 128;
  unsigned char y = 2;
  unsigned char z = (x * y) / y;
  printf("(%d * %d) / %d = %d\n",x,y,y,z);
  return 0;
}
```

À l'exécution ce code affiche `(128 * 2)/ 2 = 128` – c'est triste d'arriver à s'en émouvoir, mais c'est ainsi – car il n'y a pas débordement
Par contre si on calcule `z` en deux fois il vaudra bien 0

[C] Un développeur averti en vaut deux (2/2)

Pour ceux qui pensent que le résultat précédent résulte d'une optimisation du compilateur, voici l'assembleur généré

Source (c/castregister.asm)

```
    unsigned char x = 128;
          movb    $-128, -19(%rbp)
    unsigned char y = 2;
          movb    $2, -18(%rbp)
    unsigned char z = (x * y) / y;
          movzbl  -19(%rbp), %edx
          movzbl  -18(%rbp), %eax
          imull   %edx, %eax
          movzbl  -18(%rbp), %edx
```

Les opérations sont effectuées sur des registres 32 bits (promotion), sans prendre en compte le type des opérandes

[C] *Cast*astrophes

Ces mécanismes peuvent rendre certains contrôles caduques

Source (c/castbound.c)

```
#include <stdio.h>

void write(int tab[],int size,signed char ind,int val) {
  if (ind<size) tab[ind]=val;
  else printf("Fail\n");
}

int main(void) {
  size_t size=120; int tab[size];
  write(tab,size,127,42);
  write(tab,size,128,42);
  return 0;
}
```

[C] *Cast*astrophes

Ces mécanismes peuvent rendre certains contrôles caduques

Source (c/castbound.c)

```
#include <stdio.h>

void write(int tab[],int size,signed char ind,int val) {
  if (ind<size) tab[ind]=val;
  else printf("Fail\n");
}

int main(void) {
  size_t size=120; int tab[size];
  write(tab,size,127,42);
  write(tab,size,128,42);
  return 0;
}
```

Aucun *warning*, le première écriture est refusée (`Ko`) mais la seconde acceptée (`Ok`) et réalisée – le où est laissé à la sagacité de l'auditoire.
Question bonus : si `size=150`, que se passe-t-il ?

[C] Les derniers seront les premiers (1/2)

Encore un détail : qu'affiche le code suivant ?

Source (c/3264.c)

```
unsigned long setBit1(int n)
{ return (1 << n); }

unsigned long setBit2(int n)
{ return ((unsigned long) 1 << n); }

void main () {
  printf ("%lu\n", setBit1 (33));
  printf ("%lu\n", setBit2 (33));
}
```

[C] Les derniers seront les premiers (1/2)

Encore un détail : qu'affiche le code suivant ?

Source (c/3264.c)

```
unsigned long setBit1(int n)
{ return (1 << n); }

unsigned long setBit2(int n)
{ return ((unsigned long) 1 << n); }

void main () {
  printf ("%lu\n", setBit1 (33));
  printf ("%lu\n", setBit2 (33));
}
```

Réponse : ça dépend de l'architecture retenue pour la compilation

- En 32 bits, les deux lignes affichent 2
- En 64 bits, la première affiche 2 et la seconde 8589934592 (2^{33})

[C] Les derniers seront les premiers (2/2)

Si vous le lui demandez, le compilateur peut parfois vous aider à repérer ces problèmes (`-Wconversion` pour GCC)

Les plus curieux d'entre vous pourront étudier la faille CVE-2010-0740 (*Record of death vulnerability*) sur OpenSSL

Source (c/patch-CVE-2010-0740.diff)

```
-   /* Send back error using their
-    * version number :-) */
-   s->version=version;
+   if ((s->version & 0xFF00) == (version & 0xFF00))
+   /* Send back error using their minor version number :-) */
+   s->version = (unsigned short)version;
```

[Java] Surcharge

Allez, un petit dernier pour la route, même si à ce stade il faut bien admettre que l'effet de surprise tend à s'estomper

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

Allez, un petit dernier pour la route, même si à ce stade il faut bien admettre que l'effet de surprise tend à s'estomper

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`

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; //"`...

[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);;
```

[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 -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 (php/injectioneval-old.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 (php/injectioneval-old.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

[Php] Évaluateurs masqués. . .

D'autres constructions, sans être des évaluateurs, sont aussi inquiétantes (ne parlons même pas de la lisibilité ou de la traçabilité)

Source (php/injectionvar.php)

```
function hello() { echo "Hello world<br />"; }
function goodbye() { echo "Good bye<br />"; }

$x="hello"; $x();

$y="x"; $$y="goodbye"; $x();
```

[PHP] Évaluateurs masqués. . .

D'autres constructions, sans être des évaluateurs, sont aussi inquiétantes (ne parlons même pas de la lisibilité ou de la traçabilité)

Source (php/injectionvar.php)

```
function hello() { echo "Hello world<br />"; }
function goodbye() { echo "Good bye<br />"; }

$x="hello"; $x();

$y="x"; $$y="goodbye"; $x();
```

```
Hello world
Good bye
```

[*Shell*] *Star wars*

Le *Shell* permet d'utiliser des caractères tels que `*` dans une ligne de commande ; il y a alors substitution par l'ensemble des noms de fichiers du répertoire courant

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 `*` ?
- Un fichier nommé `-o` ?
- Un fichier nommé `> rights.acl` ?
- Un fichier nommé `; rm *` ?

[*Shell*] *Star wars*

Le *Shell* permet d'utiliser des caractères tels que `*` dans une ligne de commande ; il y a alors substitution par l'ensemble des noms de fichiers du répertoire courant

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 `*` ?
- Un fichier nommé `-o` ?
- Un fichier nommé `> rights.acl` ?
- Un fichier nommé `; rm *` ?

Tous ces cas sont possibles. . . et sans effet particulier, à l'exception du second puisqu'un fichier nommé `-o` sera généralement interprété par la commande comme une option

Plan

[Python] Localité en fête 1/3

Python offre une construction syntaxique proche du `map` classique sur les listes, la définition de liste 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 1/3

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

[Python] Localité en fête 2/3

Encore un comportement intriguant

Source (python/localvar3.py)

```
source = [1,2,3,4]
dest = [0] * 4

def copy1 ():
    dest = source

def copy2 ():
    for i in range(4):
        dest[i]=source[i]
```

[Python] Localité en fête 2/3

Encore un comportement intriguant

Source (python/localvar3.py)

```
source = [1,2,3,4]
dest = [0] * 4

def copy1 ():
    dest = source

def copy2 ():
    for i in range(4):
        dest[i]=source[i]
```

Initialement, `dest` vaut `[0,0,0,0]`, après l'appel de `copy1 ()` il vaut toujours `[0,0,0,0]` et après l'appel de `copy2 () [1,2,3,4]`

[PHP] Localité en fête 3/3

Scripts et localité ne semblent pas faire bon ménage

Source (php/link.php)

```
$var = "var";
$tab = array("foo ", "bar ", "blah ");

echo "Tab="; echo $tab; echo " ; ";
echo "Loop="; { foreach ($tab as $var) { echo $var; } }
echo " ; ";

echo "Var="; echo $var;
```

[PHP] Localité en fête 3/3

Scripts et localité ne semblent pas faire bon ménage

Source (php/link.php)

```
$var = "var";
$tab = array("foo ", "bar ", "blah ");

echo "Tab="; echo $tab; echo " ; ";
echo "Loop="; { foreach ($tab as $var) { echo $var; } }
echo " ; ";

echo "Var="; echo $var;
```

On obtient `Tab=Array ; Loop=foo bar blah ; Var=blah`, la variable `var` est écrasée et survit à la boucle ! Les plus audacieux pourront tester `foreach ($tab as $tab)...`

Plan

[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[5] ; accessoirement il peut devenir difficile de mettre en commentaire un morceau de code

Source (ocaml/errcomments.ml)

```
let x' _ _ _ _ = print_endline "non";;
let g _ = print_endline "oui";;
x'"'" "*) g" "(*" '"';;
```

Ce code, compilé et exécuté, affiche `non` ; si la dernière ligne est mise entre `(*` et `*)`, il affiche `oui`

5. C'est par exemple le cas de Coq et CoqIde

[C] Toujours pas de commentaires

Les commentaires semblent délicats à traiter dans d'autres langages

Source (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`) on obtient `2`

[C] Complément d'information

Combien y a-t-il de solutions pour l'équation $x = -x$ en $\mathbb{C}$?

[C] Complément d'information

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)
-2147483648==-(-2147483648)
```

[OCaml] Certains sont plus égaux que d'autres

Comme tout le monde le sait, un simple booléen de la forme `x=y` (avec `x` et `y` des variables) peut prendre. . . 4 valeurs !

[OCaml] Certains sont plus égaux que d'autres

Comme tout le monde le sait, un simple booléen de la forme `x=y` (avec `x` et `y` des variables) peut prendre. . . 4 valeurs !

Source (ocaml/bool4-old.ml)

```
let one=1;;

let two=2;;

type tree=Leaf | Node of tree*tree;;
let rec crash=Node(crash,crash);;

let rec loop=1::loop;;
```

`one=one` donne `true` et `one=two` donne `false`, alors que `crash=crash` plante et `loop=loop` boucle

Comment doit-on enseigner la programmation de systèmes critiques ?

[SQL] Ordre et transgressions

Alors même que tout est logique, certains comportements peuvent apparaître comme inattendus, comme dans le code suivant

Source (sql/partial.sql)

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

[SQL] Ordre et transgressions

Alors même que tout est logique, certains comportements peuvent apparaître comme inattendus, comme dans le code suivant

Source (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;` nous obtenons bien `X<=Y and X<Y`, mais avec `SET @X=NULL` le résultat est `X>Y and X<Y` : l'ordre de SQL est partiel. . . Rien de grave, mais c'est suffisamment inhabituel pour que cela mène à de faux raisonnements

[Shell] Star wars

Les gags sur les booléens à 3 valeurs (`true`, `false`, `file_not_found`) sont récurrents sur `www.thedailywtf.com` ; et pourtant...

Source (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] Star wars

Les gags sur les booléens à 3 valeurs (`true`, `false`, `file_not_found`) sont récurrents sur `www.thedailywtf.com` ; et pourtant. . .

Source (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

[JavaScript] Offusquez-vous !

La spécification JavaScript décrit de manière très précise le sens des lignes suivantes. . . qui sont tous différents, même pour la première et la dernière ligne

Source (js/weirdeval.js)

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

Le site `jscert.org` contient d'autres éléments surprenants[6] sur JavaScript. . . et des travaux sur la formalisation du langage !

6. Le qualificatif est aimable

Plan

[B] La croisière s'amuse 1/2

Quelles garanties peuvent apporter les méthodes formelles ?

Source (b/airlock.b)

```
MACHINE Airlock

VARIABLES door1,door2
INVARIANT door1,door2:{open,locked} &
          ~(door1=open & door2=open)

INITIALISATION door1:=locked || door2:=locked
OPERATIONS
  open1 = IF door2=locked THEN door1:=open
  close1 = door1:=locked
  open2 = IF door1=locked THEN door2:=open
  close2 = door2:=locked
```

[B] La croisière s'amuse 1/2

Quelles garanties peuvent apporter les méthodes formelles ?

Source (b/airlock.b)

```
MACHINE Airlock

VARIABLES door1,door2
INVARIANT door1,door2:{open,locked} &
          ~(door1=open & door2=open)

INITIALISATION door1:=locked || door2:=locked
OPERATIONS
  open1 = IF door2=locked THEN door1:=open
  close1 = door1:=locked
  open2 = IF door1=locked THEN door2:=open
  close2 = door2:=locked
```

N'embarquez pas trop vite : un développeur malicieux peut ouvrir les deux portes tout en prouvant la conformité de son implémentation

[B] La croisière s'amuse 2/2

Ce n'est pas lié à une erreur dans la méthode formelle B mais à la sémantique du langage et de ses preuves ; en effet la preuve ne garantit pas que l'invariant est toujours vrai mais

- Qu'il est vrai à l'initialisation
- Que s'il est vrai avant l'appel d'une opération, alors il reste vrai après son exécution

Rien n'est dit sur les états intermédiaires ; une implémentation ouvrant les deux portes avant de les refermer est donc conforme

Au passage, le test automatisé reposant sur un oracle ne révèlera pas forcément ce type de problème ; par exemple pour un circuit électronique dont la sortie est validée par un signal `ack`, les états transitoires ne sont absolument pas considérés. . .

[Coq] *Vanitas, vanitas*

Des problèmes peuvent aussi venir d'une incohérence entre la théorie formelle et la concrétisation par traduction dans un langage compilable

Source (coq/emptyfalse.v)

```
Inductive Empty := onemore:Empty->Empty.

Theorem emptyfalse : forall (e:Empty), False.
Proof.
  intro e; induction e as [_ He]; apply He.
Qed.
```

[Coq] *Vanitas, vanitas*

Des problèmes peuvent aussi venir d'une incohérence entre la théorie formelle et la concrétisation par traduction dans un langage compilable

Source (coq/emptyfalse.v)

```
Inductive Empty := onemore:Empty->Empty.

Theorem emptyfalse : forall (e:Empty), False.
Proof.
  intro e; induction e as [_ He]; apply He.
Qed.
```

Si le type `empty` est vide en Coq, sa traduction en OCaml `type empty = | Onemore` ne l'est pas ; l'incohérence concerne également les valeurs rationnelles d'OCaml qui sont "hors modèle" en Coq

Plan

[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*

[C] Pointless pointer operations

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

The meaning of "adding 1 to a pointer," and by extension, all pointer arithmetic, is that pa+1 points to the next object, and pa+i points to the i-th object beyond pa.

En d'autres termes, l'effet de `p++` dépend du type pointé, et plus exactement de la taille de sa représentation mémoire – qui peut être obtenue notamment par l'utilisation de l'opérateur `sizeof` ; c'est relativement intuitif

[C] Pointless pointer operations

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

The meaning of "adding 1 to a pointer," and by extension, all pointer arithmetic, is that pa+1 points to the next object, and pa+i points to the i-th object beyond pa.

En d'autres termes, l'effet de `p++` dépend du type pointé, et plus exactement de la taille de sa représentation mémoire – qui peut être obtenue notamment par l'utilisation de l'opérateur `sizeof` ; c'est relativement intuitif

Ceci dit que se passe-t-il exactement si `p` est un pointeur de fonction en architecture CISC ? En fait, cette question est dénuée de sens ; pourtant, le code correspondant est compilable et exécutable...

[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 [7] 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.*

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

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

Sur http://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 ?*

[*Shell*] État d'esprit

Une petite question anodine utilisée pour des évaluations informelles, qui met bien en évidence l'existence de différents modes de pensée. . .

Parmi les commandes suivantes, lesquelles sont susceptibles (sans redirection) de provoquer la destruction de données d'un fichier ?

`ls` ☐ `cd` ☐ `cp` ☐ `cat` ☐ `rm` ☐ `mv` ☐

Le plus souvent, la seule réponse validée est `rm`

Plan

A propos de l'enseignement

Comment former un développeur ou un évaluateur de sécurité ?

- 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 imaginer 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 [8]
- *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"

8. 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 [9]
  - Modèle mémoire reflétant l'encapsulation
  - Surveiller le flot d'exécution même en présence de fautes
- Disposer d'outils maîtrisés voire de confiance

9. 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 ;-)

Plan

Outils et langages utilisés

- B
- C : `gcc (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3`
- C++
- Coq : `The Coq Proof Assistant, version 8.3pl4 (April 2012)`
- 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)`

Pour ceux qui en veulent plus

JAVA En jouant avec les méthodes virtuelles, de l'héritage, il peut devenir très difficile de savoir ce qui va s'exécuter

JAVA Certains aspects du typage sont subtils, par exemple déclarer un tableau A a[...] puis y insérer des valeurs new B(), qui est qui ?

C++ Les abus des *Templates* peuvent nuire à la santé mentale