Obfuscation du logiciel : brouiller le code pour protéger les programmes

Sandrine Blazy

Collège de France, 2022-04-14

Obfuscation ? Brouillage ?

Obscurcir, brouiller, offusquer, obfusquer ?

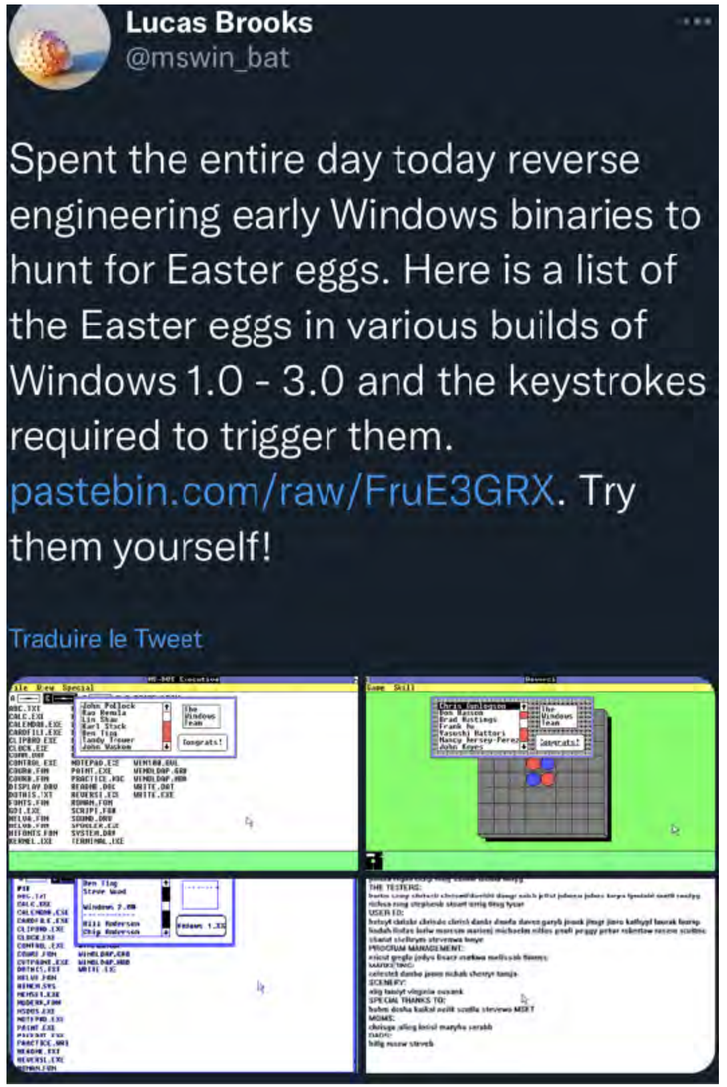
Offusquer (https://www.cnrtl.fr/etymologie/offusquer)

- Emprunté au latin *offuscare* (*fuscare* : assombrir)
- (XV^{e}) empêcher quelque chose d'être vu en le masquant
- (XVIII^{e}) porter ombrage à quelqu'un, l'indisposer

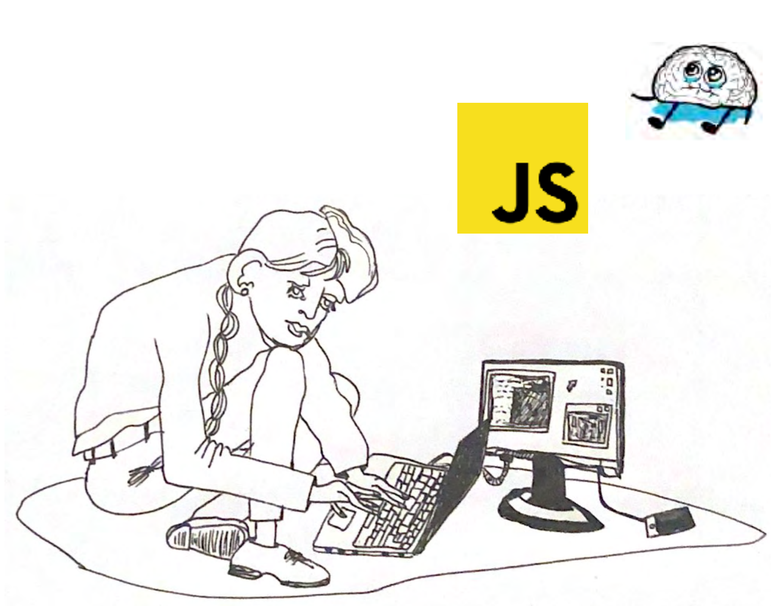
→ obscurcir

En anglais « obfuscate » (Cambridge dictionary) :
embrouiller pour cacher la vérité, rendre quelque chose moins clair et plus difficile à comprendre, notamment de manière intentionnelle
→ brouiller

Un exemple historique d'obfuscation du logiciel



Brouiller le code pour protéger les programmes
Modèle d’attaquant

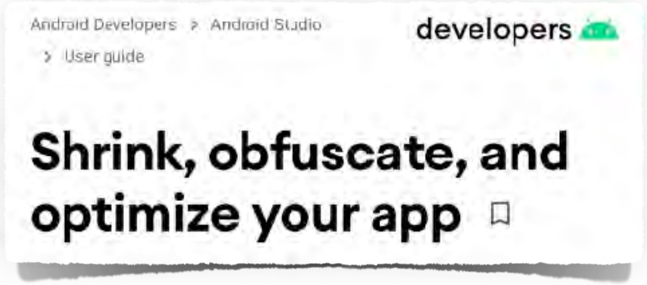


Brouiller le code pour protéger les programmes
Autres exemples

Protéger la propriété intellectuelle d’un logiciel

Dispositifs de gestion des droits numériques

Une pratique recommandée



Brouiller des programmes C

```
#define _ -F<00||--F-OO--;
int F=00,OO=00;main(){F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()
{
              _-_-_-_
         _-_-_-_-_-_-_-_-_
      _-_-_-_-_-_-_-_-_-_-_-_
   _-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
 _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_
  _-_-_-_-_-_-_-_-_-_-_-_-_-_-_
    _-_-_-_-_-_-_-_-_-_-_-_-_-_
        _-_-_-_-_-_-_-_-_
            _-_-_-_-_
}
```

(Brian Westley, IOCCC 1988)

(Calcule une valeur approchée de π)

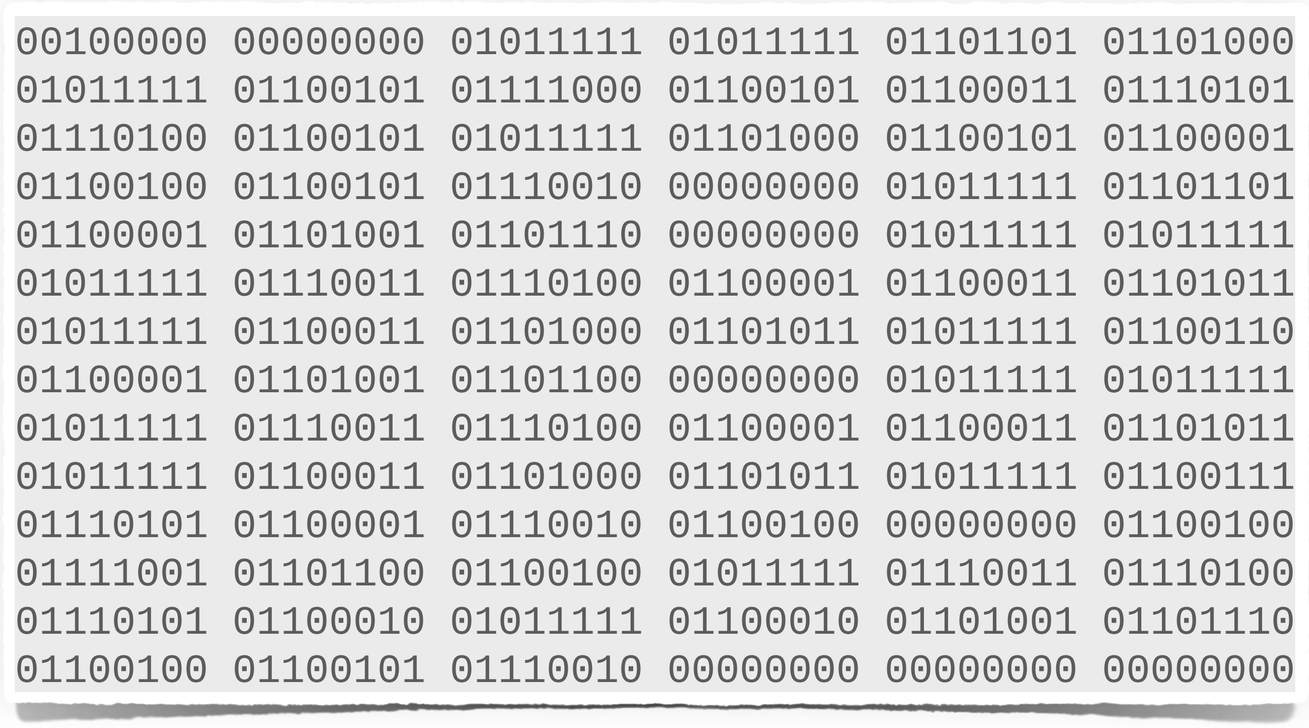
(→ cours du 28/11/2019 :
« Sémantiques mécanisées : quand la machine raisonne sur ses langages »)

Brouiller des programmes JavaScript

jscrambler

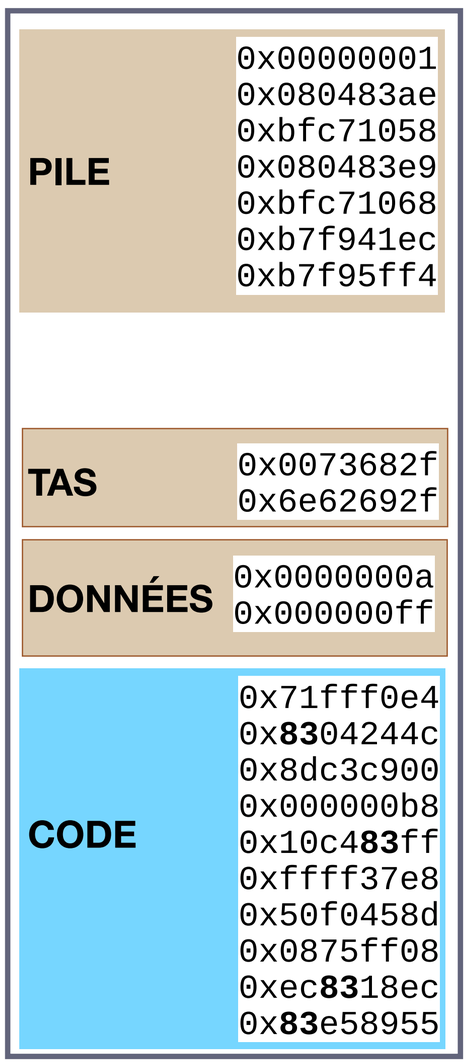
Brouiller des programmes en langage d’assemblage M/o/Vfuscator

Analyse de code en langage machine

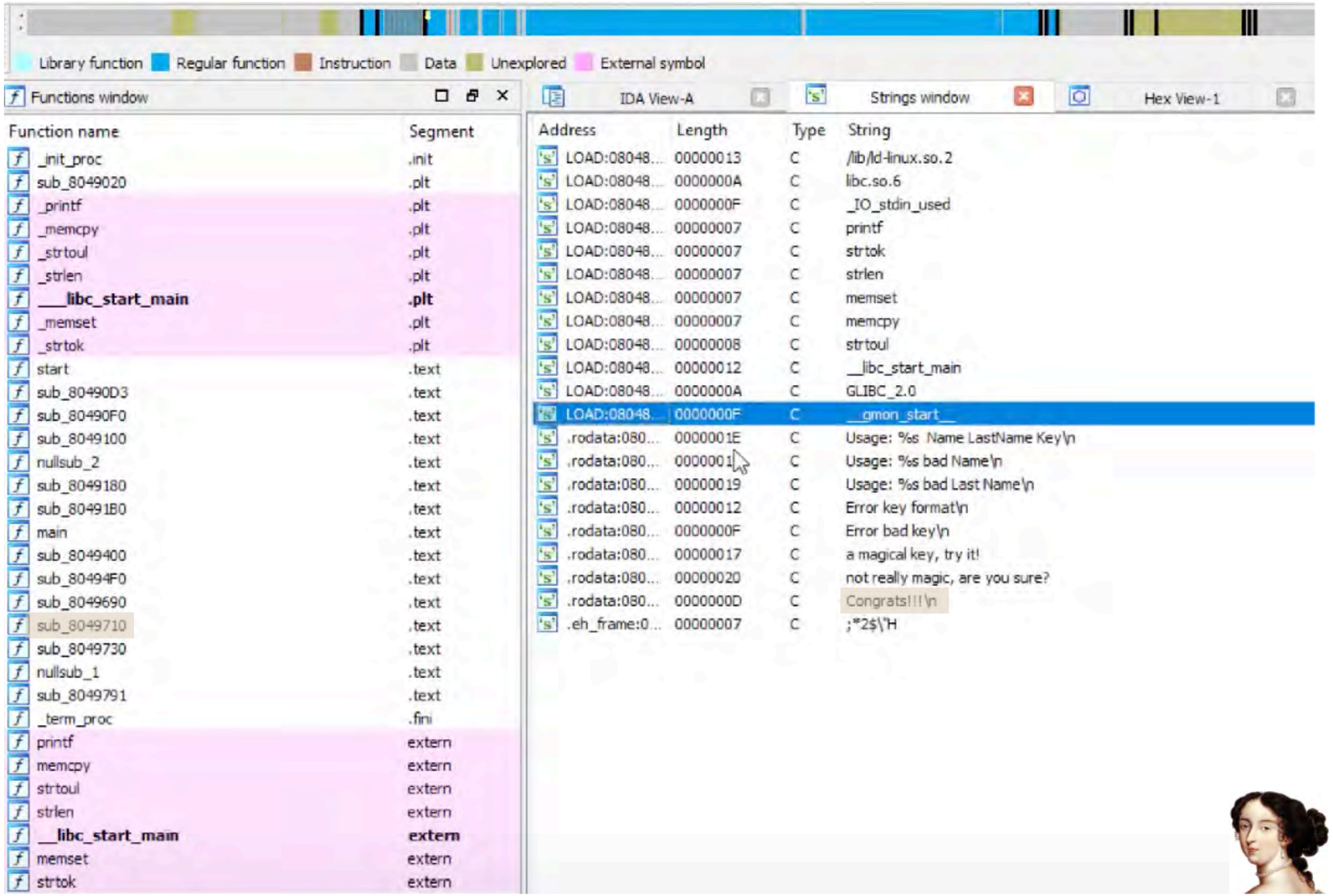


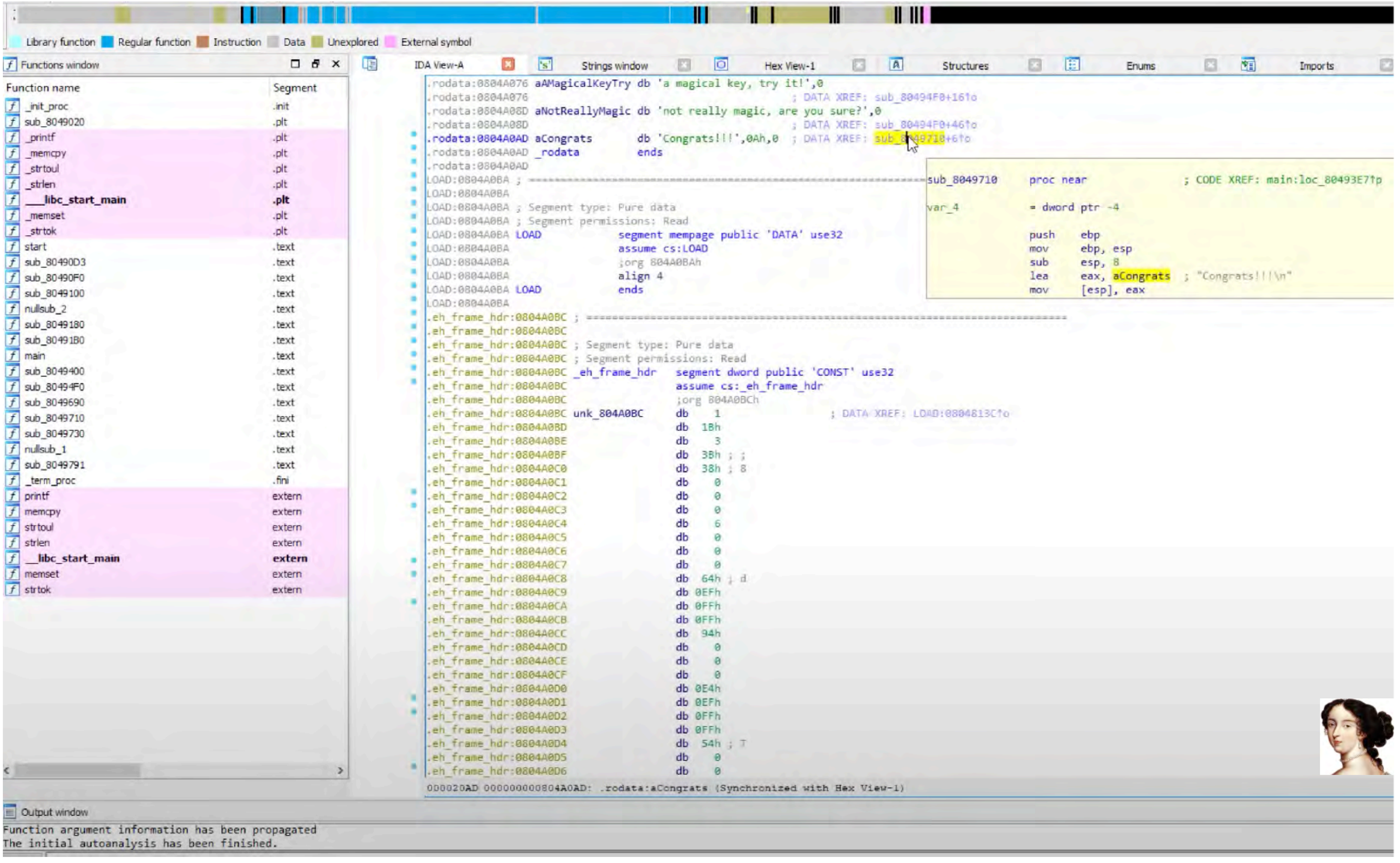
Rétro-ingénierie de code
Décodage, désassemblage

```
080483b4 <mafonction>:
80483b4:  55                    push   %ebp
80483b5:  89 e5                 mov    %esp,%ebp
80483b7:  83 ec 18              sub    $0x18,%esp
80483ba:  83 ec 08              sub    $0x8,%esp
80483bd:  ff 75 08              pushl  0x8(%ebp)
80483c0:  8d 45 f0              lea    -0x10(%ebp),%eax
80483c3:  50                    push   %eax
80483c4:  e8 37 ff ff ff        call   8048300 <strcpy@plt>
80483c9:  83 c4 10              add    $0x10,%esp
80483cc:  b8 00 00 00 00        mov    $0x0,%eax
80483d1:  c9                    leave
80483d2:  c3                    ret
080483d3 <main>:
80483d3:  8d 4c 24 04           lea    0x4(%esp),%ecx
80483d7:  83 e4 f0              and    $0xfffffff0,%esp
80483da:  ff 71 fc              pushl  -0x4(%ecx)
80483dd:  55                    push   %ebp
80483de:  89 e5                 mov    %esp,%ebp
80483e0:  51                    push   %ecx
80483e1:  81 ec 14 01 00 00     sub    $0x114,%esp
```

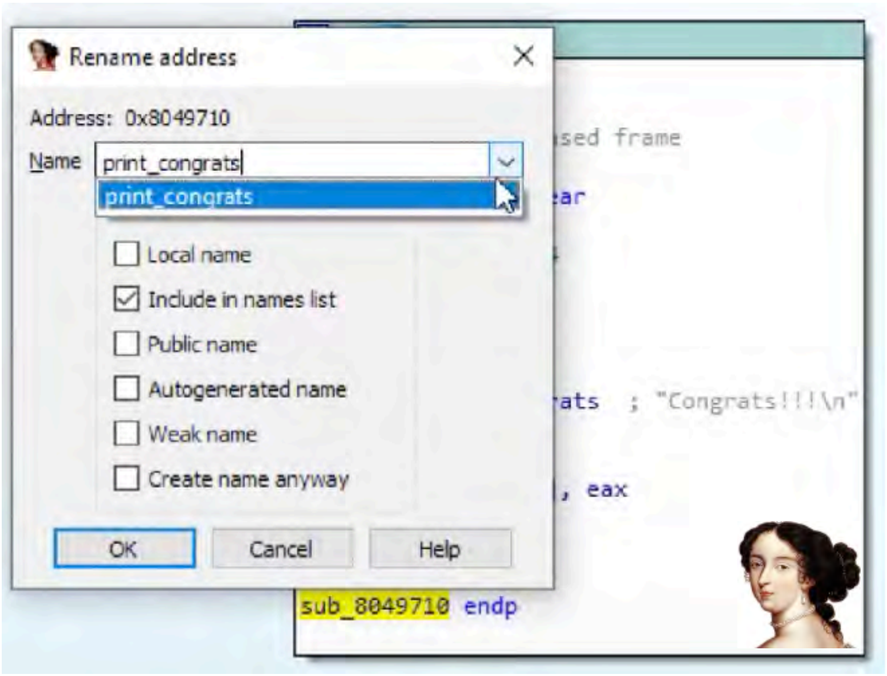


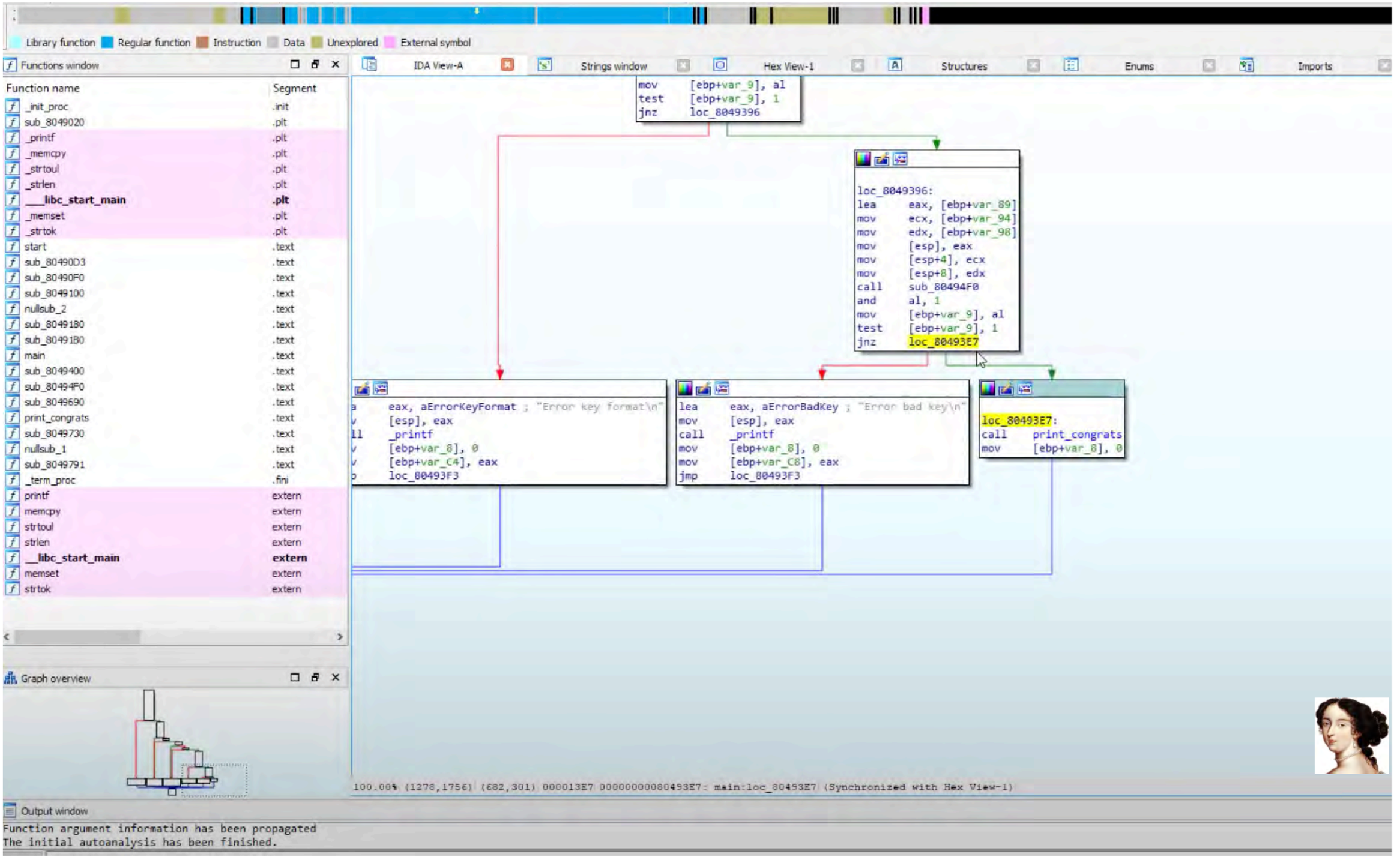
Rétro-ingénierie de code
Interagir avec un désassembleur




Library function
Regular function
Instruction
Data
Unexplored
External symbol
Functions window
Function name
Segment
_init_proc .init
sub_8049020 .plt
_printf .plt
_memcpy .plt
_strtoul .plt
_strlen .plt
__libc_start_main .plt
_memset .plt
_strtok .plt
start .text
sub_80490D3 .text
sub_80490F0 .text
sub_8049100 .text
nullsub_2 .text
sub_8049180 .text
sub_80491B0 .text
main .text
sub_8049400 .text
sub_80494F0 .text
sub_8049690 .text
sub_8049710 .text
sub_8049730 .text
nullsub_1 .text
sub_8049791 .text
_term_proc .fini
printf extern
memcpy extern
strtoul extern
strlen extern
__libc_start_main extern
memset extern
strtok extern
IDA View-A
Strings window
Hex View-1
Structures
Enums
Imports
.rodata:0804A076 aAMagicalKeyTry db 'a magical key, try it!',0
.rodata:0804A076 ; DATA XREF: sub_80494F0+16↑o
.rodata:0804A08D aNotReallyMagic db 'not really magic, are you sure?',0
.rodata:0804A08D ; DATA XREF: sub_80494F0+46↑o
.rodata:0804A0AD aCongrats db 'Congrats!!!',0Ah,0 ; DATA XREF: sub_8049710+6↑o
.rodata:0804A0AD _rodata ends
.rodata:0804A0AD
LOAD:0804A0BA ; ===========
LOAD:0804A0BA
LOAD:0804A0BA ; Segment type: Pure data
LOAD:0804A0BA ; Segment permissions: Read
LOAD:0804A0BA LOAD segment mempage public 'DATA' use32
LOAD:0804A0BA assume cs:LOAD
LOAD:0804A0BA ;org 804A0BAh
LOAD:0804A0BA align 4
LOAD:0804A0BA LOAD ends
LOAD:0804A0BA
sub_8049710 proc near ; CODE XREF: main:loc_80493E7↑p
var_4 = dword ptr -4
push ebp
mov ebp, esp
sub esp, 8
lea eax, aCongrats ; "Congrats!!!\n"
mov [esp], eax
.eh_frame_hdr:0804A0BC ; ===========
.eh_frame_hdr:0804A0BC
.eh_frame_hdr:0804A0BC ; Segment type: Pure data
.eh_frame_hdr:0804A0BC ; Segment permissions: Read
.eh_frame_hdr:0804A0BC _eh_frame_hdr segment dword public 'CONST' use32
.eh_frame_hdr:0804A0BC assume cs:_eh_frame_hdr
.eh_frame_hdr:0804A0BC ;org 804A0BCh
.eh_frame_hdr:0804A0BC unk_804A0BC db 1 ; DATA XREF: LOAD:0804813C↑o
.eh_frame_hdr:0804A0BD db 1Bh
.eh_frame_hdr:0804A0BE db 3
.eh_frame_hdr:0804A0BF db 3Bh ; ;
.eh_frame_hdr:0804A0C0 db 38h ; 8
.eh_frame_hdr:0804A0C1 db 0
.eh_frame_hdr:0804A0C2 db 0
.eh_frame_hdr:0804A0C3 db 0
.eh_frame_hdr:0804A0C4 db 6
.eh_frame_hdr:0804A0C5 db 0
.eh_frame_hdr:0804A0C6 db 0
.eh_frame_hdr:0804A0C7 db 0
.eh_frame_hdr:0804A0C8 db 64h ; d
.eh_frame_hdr:0804A0C9 db 0EFh
.eh_frame_hdr:0804A0CA db 0FFh
.eh_frame_hdr:0804A0CB db 0FFh
.eh_frame_hdr:0804A0CC db 94h
.eh_frame_hdr:0804A0CD db 0
.eh_frame_hdr:0804A0CE db 0
.eh_frame_hdr:0804A0CF db 0
.eh_frame_hdr:0804A0D0 db 0E4h
.eh_frame_hdr:0804A0D1 db 0EFh
.eh_frame_hdr:0804A0D2 db 0FFh
.eh_frame_hdr:0804A0D3 db 0FFh
.eh_frame_hdr:0804A0D4 db 54h ; T
.eh_frame_hdr:0804A0D5 db 0
.eh_frame_hdr:0804A0D6 db 0
000020AD 000000000804A0AD: .rodata:aCongrats (Synchronized with Hex View-1)
Output window
Function argument information has been propagated
The initial autoanalysis has been finished.

Rétro-ingénierie de code
Interagir avec un désassembleur




Library function
Regular function
Instruction
Data
Unexplored
External symbol
Functions window
Function name
Segment
_init_proc .init
sub_8049020 .plt
_printf .plt
_memcpy .plt
_strtoul .plt
_strlen .plt
___libc_start_main .plt
_memset .plt
_strtok .plt
start .text
sub_80490D3 .text
sub_80490F0 .text
sub_8049100 .text
nullsub_2 .text
sub_8049180 .text
sub_80491B0 .text
main .text
sub_8049400 .text
sub_80494F0 .text
sub_8049690 .text
print_congrats .text
sub_8049730 .text
nullsub_1 .text
sub_8049791 .text
_term_proc .fini
printf extern
memcpy extern
strtoul extern
strlen extern
__libc_start_main extern
memset extern
strtok extern
IDA View-A
Strings window
Hex View-1
Structures
Enums
Imports
mov [ebp+var_9], al
test [ebp+var_9], 1
jnz loc_8049396
loc_8049396:
lea eax, [ebp+var_89]
mov ecx, [ebp+var_94]
mov edx, [ebp+var_98]
mov [esp], eax
mov [esp+4], ecx
mov [esp+8], edx
call sub_80494F0
and al, 1
mov [ebp+var_9], al
test [ebp+var_9], 1
jnz loc_80493E7
eax, aErrorKeyFormat ; "Error key format\n"
[esp], eax
_printf
[ebp+var_8], 0
[ebp+var_C4], eax
loc_80493F3
lea eax, aErrorBadKey ; "Error bad key\n"
mov [esp], eax
call _printf
mov [ebp+var_8], 0
mov [ebp+var_C8], eax
jmp loc_80493F3
loc_80493E7:
call print_congrats
mov [ebp+var_8], 0
Graph overview
100.00% (1278,1756) (682,301) 000013E7 00000000080493E7: main:loc_80493E7 (Synchronized with Hex View-1)
Output window
Function argument information has been propagated
The initial autoanalysis has been finished.

IDA View-A | Stack of main | Strings window | Hex View-1 | S

```
-0000000000000088          db ? ; undefined
-0000000000000089          db ?
-0000000000000088          db ? ; undefined
-0000000000000087          db ? ; undefined
-0000000000000086          db ? ; undefined
-0000000000000085          db ? ; undefined
-0000000000000084          db ? ; undefined
-0000000000000083          db ? ; undefined
-0000000000000082          db ? ; undefined
-0000000000000081          db ? ; undefined
-0000000000000080          db ? ; undefined
-000000000000007F          db ? ; undefined
-000000000000007E          db ? ; undefined
-000000000000007D          db ? ; undefined
-000000000000007C          db ? ; undefined
-000000000000007B          db ? ; undefined
-000000000000007A          db ? ; undefined
-0000000000000079          db ? ; undefined
-0000000000000078          db ? ; undefined
-0000000000000077          db ? ; undefined
-0000000000000076          db ? ; undefined
-0000000000000075          db ? ; undefined
-0000000000000074          db ? ; undefined
-0000000000000073          db ? ; undefined
-0000000000000072          db ? ; undefined
-0000000000000071          db ? ; undefined
-0000000000000070          db ? ; undefined
-000000000000006F          db ? ; undefined
-000000000000006E          db ? ; undefined
-000000000000006D          db ? ; undefined
-000000000000006C          db ? ; undefined
-000000000000006B          db ? ; undefined
-000000000000006A          db ? ; undefined
-0000000000000069          db ? ; undefined
-0000000000000068          db ? ; undefined
-0000000000000067          db ? ; undefined
-0000000000000066          db ? ; undefined
-0000000000000065          db ? ; undefined
-0000000000000064          db ? ; undefined
-0000000000000063          db ? ; undefined
-0000000000000062          db ? ; undefined
-0000000000000061          db ? ; undefined
-0000000000000060          db ? ; undefined
-000000000000005F          db ? ; undefined
-000000000000005E          db ? ; undefined
-000000000000005D          db ? ; undefined
-000000000000005C          db ? ; undefined
-000000000000005B          db ? ; undefined
-000000000000005A          db ? ; undefined
-0000000000000059          db ? ; undefined
SP+000000000000004F
```

Convert to array

Start offset : 0x4F
End offset : 0xCF

Array element size : 1
Maximal possible size: 128
Current array size : 1
Suggested array size : 128

Array size 0x80 (in elements)
Items on a line 0 (0-max)
Element print width -1 (-1-none,0-auto)

Options
- [x] Use "dup" construct
- [ ] Signed elements
- [ ] Display indexes
- [x] Create as array

Indexes
- (•) Decimal
- ( ) Hexadecimal
- ( ) Octal
- ( ) Binary

OK Cancel Help

IDA View-A | Stack of main | Strings window | Hex View-1

```
-00000000000000A8 var_A8          dd ?
-00000000000000A4 var_A4          dd ?
-00000000000000A0 len_LastName    dd ?
-000000000000009C len_Name        dd ?
-0000000000000098 key             dd ?
-0000000000000094 LastName        dd ?
-0000000000000090 name            dd ?
-000000000000008C                 db ? ; undefined
-000000000000008B                 db ? ; undefined
-000000000000008A                 db ? ; undefined
-0000000000000089 NameOnStack     db 128 dup(?)
-0000000000000009 return_value    db ?
-0000000000000008 var_8           dd ?
-0000000000000004                 db ? ; undefined
-0000000000000003                 db ? ; undefined
-0000000000000002                 db ? ; undefined
-0000000000000001                 db ? ; undefined
+0000000000000000  s              db 4 dup(?)
+0000000000000004  r              db 4 dup(?)
+0000000000000008 argv            dd ?
+000000000000000C argc            dd ?                    ; offset
+0000000000000010
+0000000000000010 ; end of stack variables
```

Analyse de code en langage machine
Bilan

Du langage machine à la reconstruction interactive du flot de contrôle

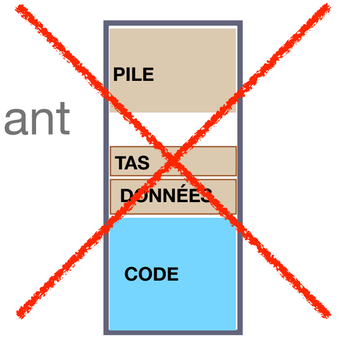
Analyse statique, dont exécution symbolique et emploi de solveurs SMT

Brouiller le code : le transformer afin de rendre plus difficile l'extraction d'informations lors de la rétro-ingénierie

« Afin d'empêcher une rétro-ingénierie fructueuse, vous devez utiliser un outil d'obfuscation. » Open Web Application Security Project (OWASP), 2016 (https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering)

Évolution concomitante des techniques de brouillage et d'analyse de code

Une parade à la rétro-ingénierie : le code auto-modifiant (ex. virtualisation d'un code)



Stratégies d’obfuscation du logiciel

(Banescu, Pretschner. A tutorial on software obfuscation. Advances in computers 2018)


```
int pc = 1;
while (pc != 0) {
  switch (pc) {
    case 1 : {
      i = 0;
      pc = 2;
      break; }
    case 2 : {
      if (i <= 100)
        pc = 3;
      else pc = 0;
      break; }
    case 3 : {
      i++;
      pc = 2;
      break; }
  } }
```

À quel niveau brouiller ?

Le langage d’assemblage, un candidat naturel

- perte des identifiants, des types, de la structure des programmes
- simple brouillage

Le langage source, pour profiter de l’abstraction

- brouiller à plus gros grain, de façon plus avancée
- davantage de possibilités de brouillage

Compilation et obfuscation

- similitude des techniques employées
- résistance aux optimisations

Obfuscation du logiciel : critères d’évaluation

Coût induit par le brouillage : conserver l’efficacité du code binaire brouillé

Résilience : difficulté à dés-obfusquer

Furtivité : s'adapter au code à protéger

Correction : préserver la sémantique des programmes

Portée : compromis entre coût et résilience
→ brouiller les parties les plus sensibles ?

1. Brouillages élémentaires

Renommer les variables

Brouiller simplement les données

- brouiller la mémoire : encoder des constantes (chaînes de caractères)
- découper des variables
- modifier la dimension d’un tableau

Brouiller les fonctions : expanser en ligne (ou l’inverse)

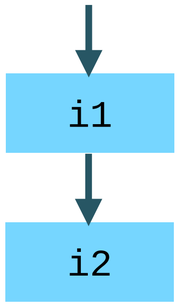
Brouiller le flot de contrôle
2. Prédicats opaques

(Collberg, Thomborson. Manufacturing cheap, resilient and stealthy opaque constructs. POPL 1998)

L'obfuscation vue comme une transformation de programme

```
int original (int n) {
return n+3;
}
```

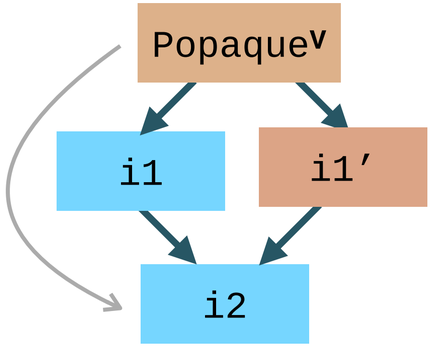
```
int brouillé (int n) {
if ((n+1)*n %2 == 0)
   return n+3;
else return n*n*n;
}
```



$$\forall x \in \mathbb{Z}, 3 \mid x^3 - x$$

$$\forall x, y \in \mathbb{Z}, x^2 - 34y^2 \neq 1$$

$$\forall x \in \mathbb{N}^+, 9 \mid 10^x + 3.4^{x+2} + 5$$



Brouiller le flot de contrôle
Prédicats opaques : variantes

Résilience : combiner avec d'autres brouillages élémentaires

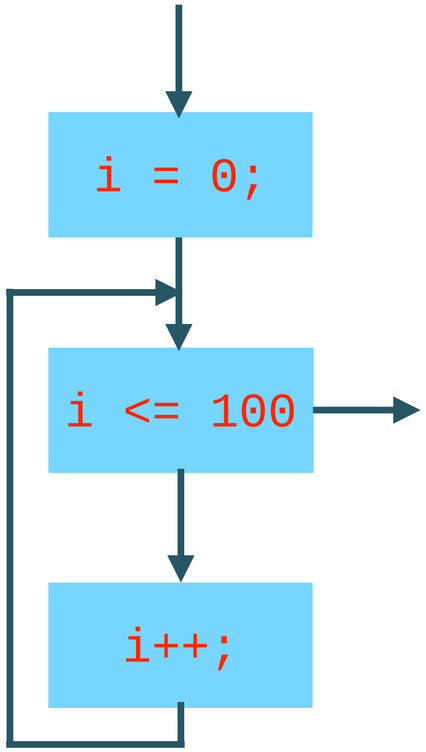
(Banescu,Collberg, Ganesh, Newsham, Pretschner. Code abfuscation against symbolic execution attacks. ACSAC 2016)

Ajouter des branchements qui dépendent des entrées et
introduire de nouveaux chemins d'exécution possibles

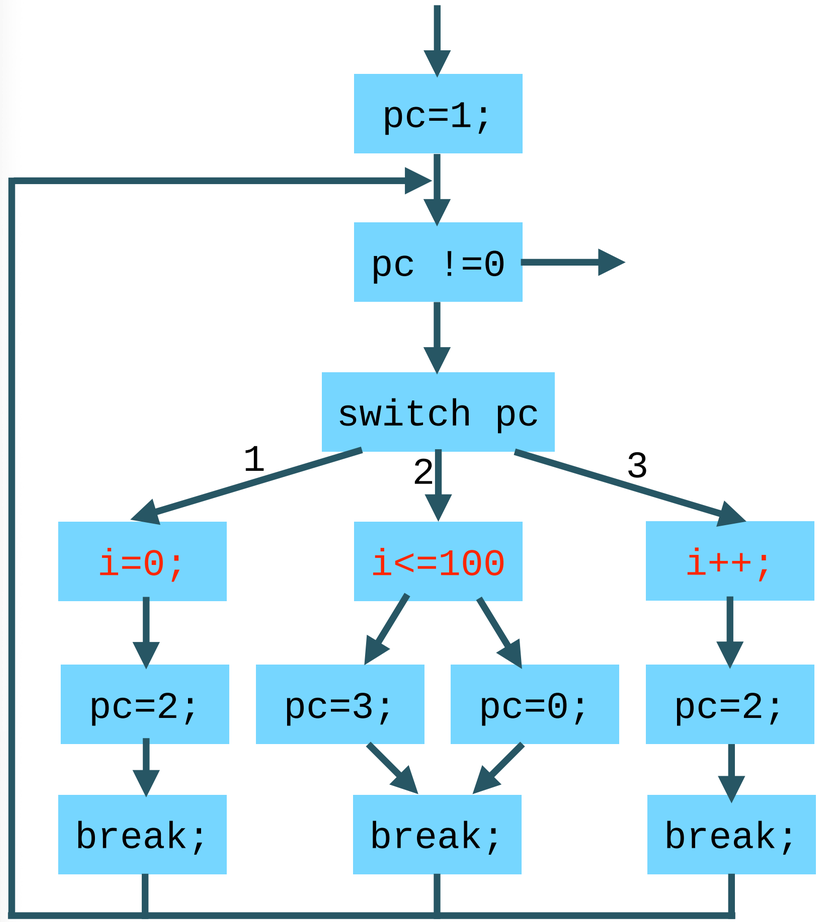
Objectif : contrer l'exécution symbolique

3. Brouiller le graphe de flot de contrôle en l'aplatissant

```
i = 0;
while (i <= 100) {
i++; }
```




```
int pc = 1;
while (pc != 0) {
  switch (pc) {
    case 1 : {
      i = 0;
      pc = 2;
      break; }
    case 2 : {
      if (i <= 100)
        pc = 3;
      else pc = 0;
      break; }
    case 3 : {
      i++;
      pc = 2;
      break; }
  }
}
```



Brouiller le flot de contrôle

Transformations présentes dans les outils d'obfuscation de l'état de l'art

- nombreux logiciels commerciaux
- Tigress (Collberg, https://tigress.wtf) - niveau C - pléthore de transformations 

- LLVM-obfuscator (Junod, Rinaldini, Wehrli, Michielin. Obfuscator-LLVM - Software Protection for the Masses. SPRO 2015) - niveau LLVM IR
- JavaScript obfuscator tool (https://obfuscator.io/) 

Connaître leurs stratégies d'obfuscation affaiblit-il la protection apportée ?

Brouiller pour diversifier le logiciel

(Cohen. Operating system protection through program evolution. Computers and security 1993)
« Stratégie du leurre »

(Pucella, Schneider. Independence from obfuscation: a semantic framework for diversity. CSF 2006)

Brouillage polymorphe : augmenter le temps passé par les attaquants, sans trop augmenter celui passé à se protéger

Diversifier pour tracer

La diversité vue comme une défense : ne pas exhiber la même vulnérabilité

- Déjà employé par les systèmes d'exploitation (ex. ASLR)
- Empêcher un attaquant d'apprendre comment le programme a été brouillé

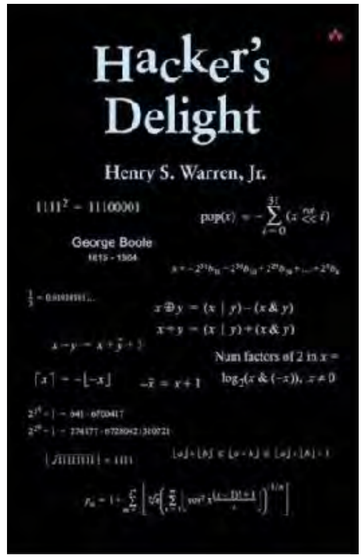
Brouiller les données
4. Expressions mixtes arithmético-booléennes

(Zhou, Main, Gu, Johnson. Information hiding in software with mixed boolean-arithmetic transforms. WISA 2007)

Opérateurs arithmétiques bit à bit de $\mathbb{Z}/(2^N)\mathbb{Z} : +, -, \times$

Opérateurs booléens de $\{0;1\}^N : \wedge, \vee, \oplus$

Mais $\begin{cases} x \times (y \wedge z) \neq (x \times y) \wedge (x \times z) \\ x \vee (y + z) \neq (x \vee y) + (x \vee z) \end{cases}$



Brouiller des expressions en mélangeant les opérateurs

- **réécrire**

  Exemple : $x + y$ devient $(x \oplus y) + 2 \times (x \wedge y)$

- **insérer des identités** $f^{-1}(f(n))$

  Exemple : $f(n) = 3 \times n + 7$ et $f^{-1}(n) = 11 \times n + 3$ dans $\mathbb{Z}/(2^4)\mathbb{Z}$

  $x + y$ devient $(((x \oplus y) + 2 \times (x \wedge y)) \times 3 + 7) \times 11 + 3$

Stratégies d’obfuscation : bilan

Renommer : rendre difficile la lecture par un humain

Ajouter des prédicats opaques : contraindre l’analyse statique à approximer davantage

Aplatir le graphe de flot de contrôle : pousser l’analyse dans ses retranchements, simuler une virtualisation

Ajouter des expressions mixtes arithmético-booléennes : déjouer les solveurs SMT, afin de les empêcher de prédire des chemins exécutables

Évaluation des deux dernières stratégies d’obfuscation

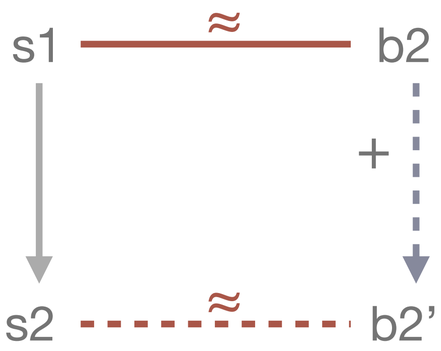
(Blazy, Trieu. Formal verification of control-flow graph flattening. CPP 2016)
(Blazy, Hutin. Formal verification of a program optimization based on mixed boolean-arithmetic expressions. CPP 2019)

Ajout d’une passe de brouillage au compilateur C CompCert (langage Clight)

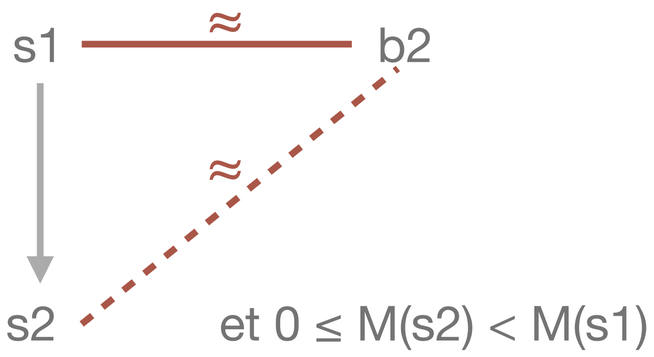
Correction : préservation sémantique, démontrée à l’aide de diagrammes de simulation relatifs aux transitions de la sémantique à continuations de Clight
→ cours du 12/12/2019 : « Vérification formelle d’un compilateur »

Diagrammes de simulation « étoile »

Chaque transition du source S est simulée par zéro, une ou plusieurs transitions dans le code brouillé B.



ou




```
i = 0;
```

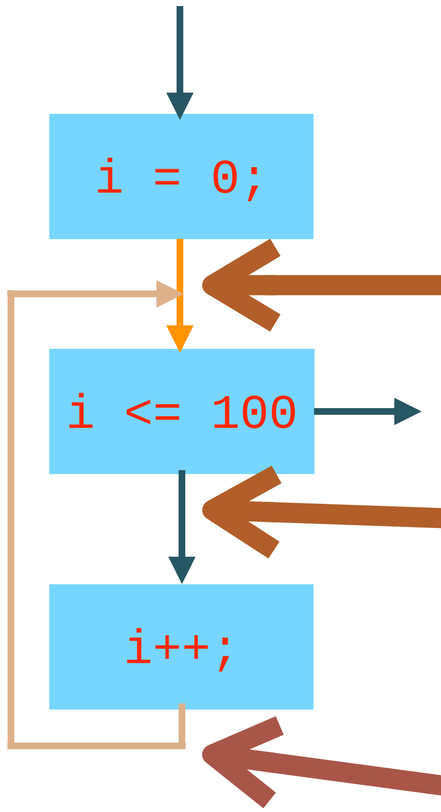
```
switch (pc) {
   case 1 : {
     i = 0;
     pc = 2;
     break; }}
```

Difficultés

- comment relier une commande de S aux commandes de B
- trouver la mesure M qui évite le bégaiement infini

Aplatissement du graphe de flot de contrôle : argument de correction

Relier une commande de S aux commandes de B : indiquer comment reconstruire le graphe de flot de contrôle de S à partir (de l'arbre de syntaxe abstraite) de B




```
int pc = 1;
while (pc != 0) {
  switch (pc) {
    case 1 : {
      i = 0;
      pc = 2; break;
      }
    case 2 : {
      if (i <= 100)
        pc = 3;
      else pc = 0;
      break; }
    case 3 : {
      i++;
      pc = 2; break;
      }
} }
```

Insertion d'expressions mixtes arithmético-booléennes : argument de correction

Comment établir que $x + y = (x \oplus y) + 2 \times (x \wedge y)$?

Une induction sur le nombre de bits ne suffit pas.

Solution : définir une structure de données généralisant la notion de table de vérité

- facilitant la preuve

→ propriété de localité

x	b	op	y	b'	=	x op y	b op b'

Évaluer indépendamment b op b', sans modifier les autres bits

- indépendante du langage considéré

Conclusion

Grande variété de techniques, assez simples à mettre en œuvre

C’est leur combinaison savamment dosée qui permet de protéger un logiciel

Évaluation par la pratique, par des experts

Une histoire sans fin, liée à l’évolution des techniques de dés-obfuscation

Une protection parmi d’autres (ex. isolation logicielle, cryptographie, tatouage numérique, « empaquetage » de logiciel), apportant un gain de temps (qui vise néanmoins à décourager un attaquant)

Questions ?