Du code à la faille de sécurité . . .

Éric BERTHOMIER
berthomiereric70@gmail.com

4 janvier 2026

Version 1.1 - Version Stagiaire

Qu'est ce qu'un code source ?

Un code source est un texte lisible et modifiable par l'être humain qui lui permet de créer un programme.

bonjour.py

```
#!/usr/bin/python3

print ("Hello World !")
```

Qu'est ce qu'un code source ?

Un code source est un texte lisible et modifiable par l'être humain qui lui permet de créer un programme.

bonjour.py

```
#!/usr/bin/python3

print ("Hello World !")
```

bonjour.c

```
#include <stdio.h>

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

Interprété

bonjour.py

```
#!/usr/bin/python3

print ("Hello World !")
```

- Ce programme est un code interprété, il est lu et exécuté au fur et à mesure par le programme Python.

Interprété

bonjour.py

```
#!/usr/bin/python3

print ("Hello World !")
```

- Ce programme est un code interprété, il est lu et exécuté au fur et à mesure par le programme Python.
- Sous Windows, le langage le plus utilisé est le PowerShell qui lui aussi est interprété.

Interprété

bonjour.py

```
#!/usr/bin/python3

print ("Hello World !")
```

- Ce programme est un code interprété, il est lu et exécuté au fur et à mesure par le programme Python.
- Sous Windows, le langage le plus utilisé est le PowerShell qui lui aussi est interprété.
- Il est possible d'analyser le code à tout moment et c'est pour cette raison que les malwares obfusquent le code interprété.

Obscurantisme

bonjour_obf.py

```
#!/bin/python3

import base64
exec(base64.b64decode("cHJpbnQoIkhlbGxvIFdvcmxkICEiKQ=="))
```

Réalise la même chose que le précédent programme.

Compilé

bonjour.c

```c
#include <stdio.h>

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

- Ce programme est un code source, il ne peut pas être exécuté tel que.

Compilé

bonjour.c

```c
#include <stdio.h>

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

- Ce programme est un code source, il ne peut pas être exécuté tel que.
- Pour être exécuter il faut le compiler : `gcc -o bonjour bonjour.c`.

Compilé

bonjour.c

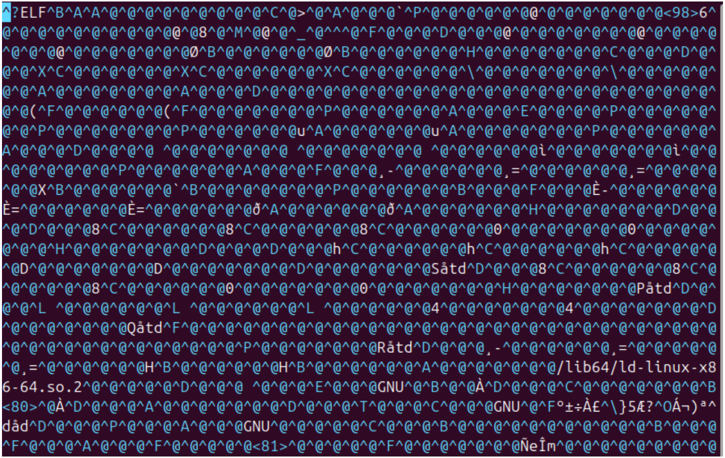
```c
#include <stdio.h>

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

- Ce programme est un code source, il ne peut pas être exécuté tel que.
- Pour être exécuter il faut le compiler : `gcc -o bonjour bonjour.c`.
- Sa forme devient alors parfaitement illisible pour l'être humain.

Exécutable

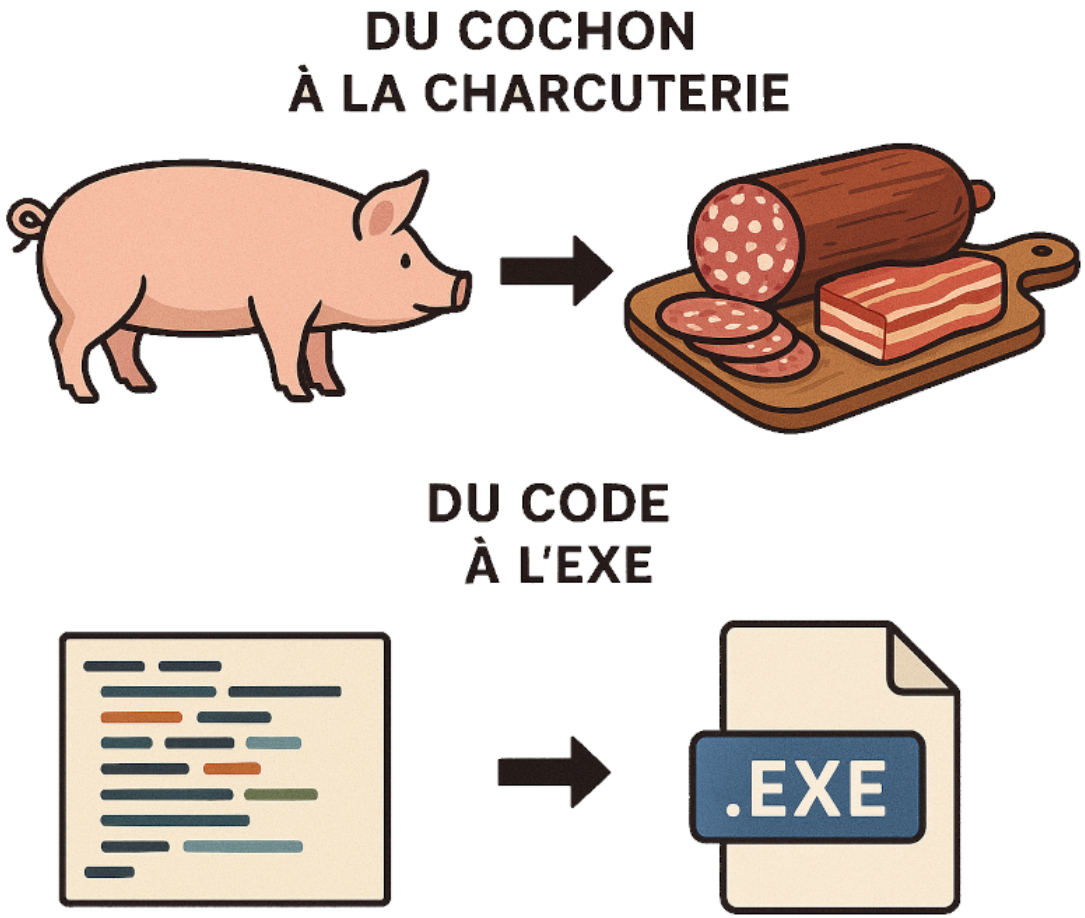

```
^?ELF^B^A^A^@^@^@^@^@^@^@^@^@^C^@>^@^A^@^@^@`^P^@^@^@^@^@^@@^@^@^@^@^@^@^@<98>6^
@^@^@^@^@^@^@^@^@^@@^@8^@^M^@@^@^_^@^^^@^F^@^@^@^D^@^@^@@^@^@^@^@^@^@^@@^@^@^@^@
^@^@^@@^@^@^@^@^@^@^@Ø^B^@^@^@^@^@^@Ø^B^@^@^@^@^@^@^H^@^@^@^@^@^@^@^C^@^@^@^D^@^
@^@^X^C^@^@^@^@^@^@^X^C^@^@^@^@^@^@^X^C^@^@^@^@^@^@^\^@^@^@^@^@^@^@^\^@^@^@^@^@^
@^@^A^@^@^@^@^@^@^@^A^@^@^@^D^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@(^F^@^@^@^@^@^@(^F^@^@^@^@^@^@^@^P^@^@^@^@^@^@^A^@^@^@^E^@^@^@^@^P^@^@^@^@^@^
@^@^P^@^@^@^@^@^@^@^P^@^@^@^@^@^@u^A^@^@^@^@^@^@u^A^@^@^@^@^@^@^@^P^@^@^@^@^@^@^
A^@^@^@^D^@^@^@^@ ^@^@^@^@^@^@^@ ^@^@^@^@^@^@^@ ^@^@^@^@^@^@ì^@^@^@^@^@^@^@ì^@^@
^@^@^@^@^@^@^P^@^@^@^@^@^@^A^@^@^@^F^@^@^@¸-^@^@^@^@^@^@¸=^@^@^@^@^@^@¸=^@^@^@^@
^@^@X^B^@^@^@^@^@^@`^B^@^@^@^@^@^@^@^P^@^@^@^@^@^@^B^@^@^@^F^@^@^@È-^@^@^@^@^@^@
È=^@^@^@^@^@^@È=^@^@^@^@^@^@ð^A^@^@^@^@^@^@ð^A^@^@^@^@^@^@^H^@^@^@^@^@^@^@^D^@^@
^@^D^@^@^@8^C^@^@^@^@^@^@8^C^@^@^@^@^@^@8^C^@^@^@^@^@^@0^@^@^@^@^@^@^@0^@^@^@^@^
@^@^@^H^@^@^@^@^@^@^@^D^@^@^@^D^@^@^@h^C^@^@^@^@^@^@h^C^@^@^@^@^@^@h^C^@^@^@^@^@
^@D^@^@^@^@^@^@^@D^@^@^@^@^@^@^@^D^@^@^@^@^@^@^@Såtd^D^@^@^@8^C^@^@^@^@^@^@8^C^@
^@^@^@^@^@8^C^@^@^@^@^@^@0^@^@^@^@^@^@^@0^@^@^@^@^@^@^@^H^@^@^@^@^@^@^@Påtd^D^@^
@^@^L ^@^@^@^@^@^@^L ^@^@^@^@^@^@^L ^@^@^@^@^@^@4^@^@^@^@^@^@^@4^@^@^@^@^@^@^@^D
^@^@^@^@^@^@^@Qåtd^F^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^P^@^@^@^@^@^@^@Råtd^D^@^@^@¸-^@^@^@^@^@^@¸=^@^@^@^@
^@¸=^@^@^@^@^@^@H^B^@^@^@^@^@^@H^B^@^@^@^@^@^@^A^@^@^@^@^@^@^@/lib64/ld-linux-x8
6-64.so.2^@^@^@^@^@^D^@^@^@ ^@^@^@^E^@^@^@GNU^@^B^@^@À^D^@^@^@^C^@^@^@^@^@^@^@^B
<80>^@À^D^@^@^@^A^@^@^@^@^@^@^@^D^@^@^@^T^@^@^@^C^@^@^@GNU^@^F°±÷À£^\}5Æ?^OÁ¬)ª^
dåd^D^@^@^@^P^@^@^@^A^@^@^@GNU^@^@^@^@^@^C^@^@^@^B^@^@^@^@^@^@^@^@^@^@^@^B^@^@^@
^F^@^@^@^A^@^@^@^F^@^@^@^@<81>^@^@^@^@^F^@^@^@^@^@^@ÑeÎm^@^@^@^@^@^@^@^@
```

Exécutable

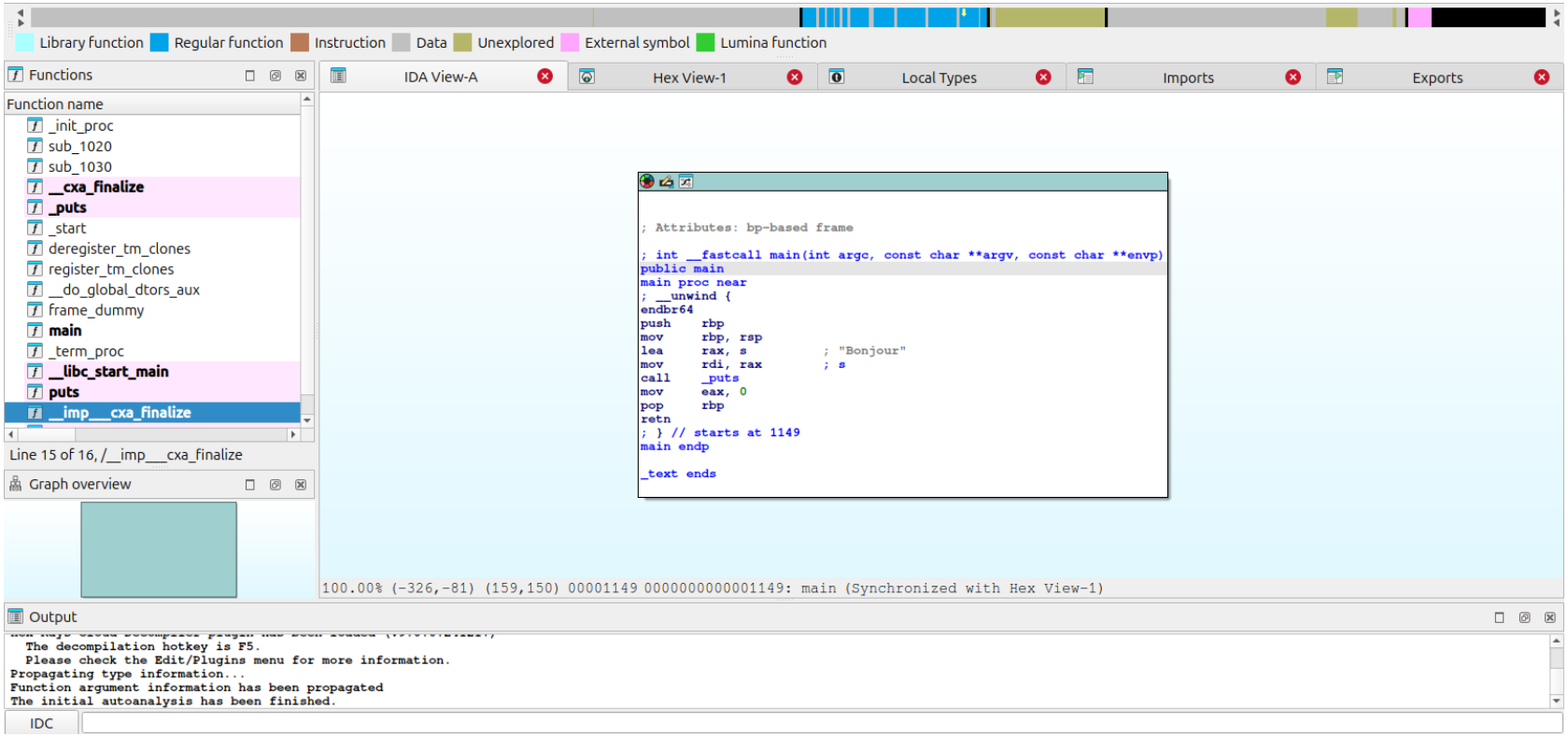


Il faut alors utiliser des outils spécifiques pour **essayer** de découvrir ce que fait le programme.

Du code à l'EXE



Désassembler le programme "bonjour"



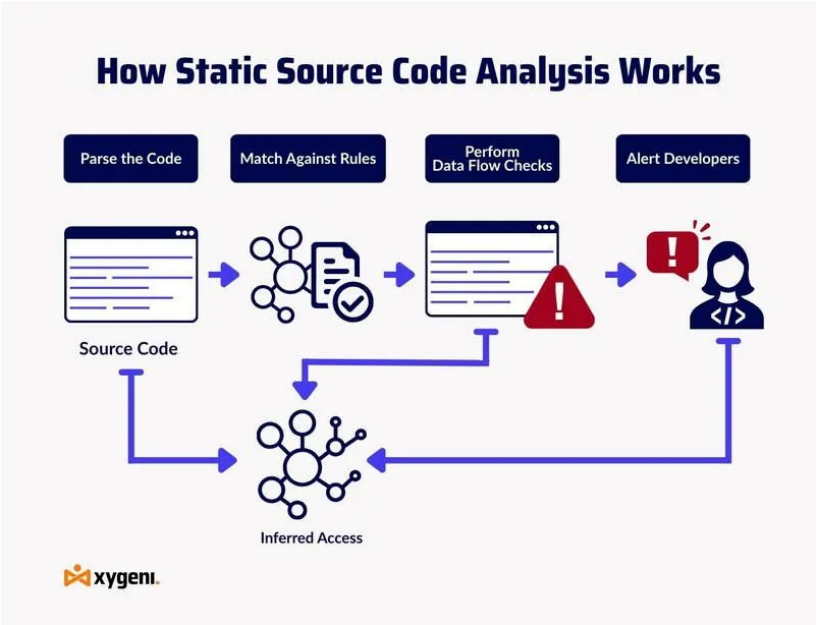
Assembleur

Générer le code "Assembleur" à partir du source : `gcc -S`.

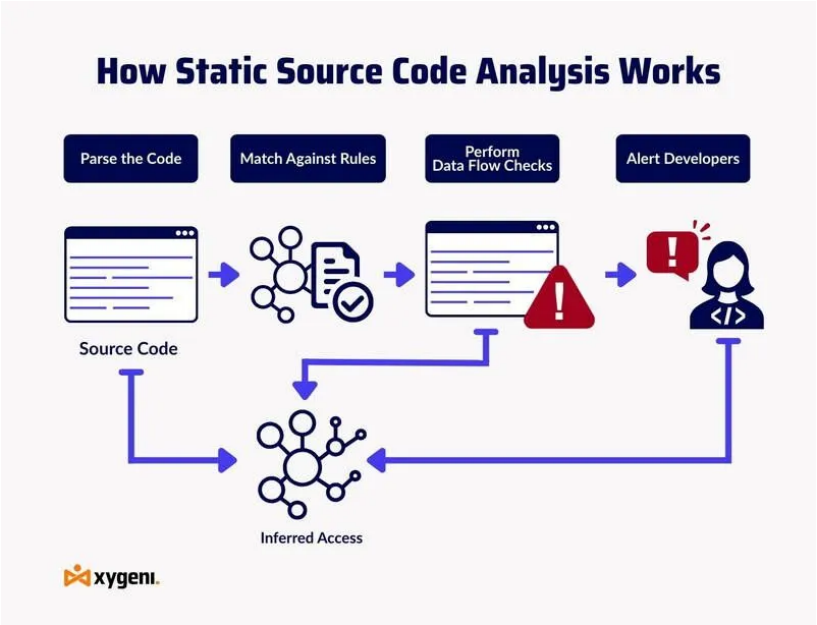
```
    .file   "bonjour.c"
    .text
    .section    .rodata
.LC0:
    .string "Bonjour"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    leaq    .LC0(%rip), %rax
    movq    %rax, %rdi
    call    puts@PLT
    movl    $0, %eax
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

```
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 13.3.0-6ubuntu2~24.04
          ) 13.3.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long   1f - 0f
    .long   4f - 1f
    .long   5
0:
    .string "GNU"
1:
    .align 8
    .long   0xc0000002
    .long   3f - 2f
2:
    .long   0x3
3:
    .align 8
4:
```

Un programme est donc analysé **lors de sa conception** à partir de son code source (qui est lisible par l'humain).



Un programme est donc analysé **lors de sa conception** à partir de son code source (qui est lisible par l'humain).



C'est ce qui est réalisé pour les applications Métier par l'équipe de Jacinthe.

SSI - Quelques raisons. . .

1. Le développeur peut faire une erreur dans son code.

SSI - Quelques raisons. . .

1. Le développeur peut faire une erreur dans son code.
2. Le développeur peut télécharger un élément malveillant qui n'aura pas été validé auparavant.

SSI - Quelques raisons. . .

1. Le développeur peut faire une erreur dans son code.
2. Le développeur peut télécharger un élément malveillant qui n'aura pas été validé auparavant.
3. Ce qui est échangé sur le réseau peut être chiffré et donc invisible.

SSI - Quelques raisons. . .

1. Le développeur peut faire une erreur dans son code.
2. Le développeur peut télécharger un élément malveillant qui n'aura pas été validé auparavant.
3. Ce qui est échangé sur le réseau peut être chiffré et donc invisible.
4. L'analyste ne peut pas analyser tous les programmes et **doit pouvoir faire confiance** (whitelist).

SSI - Quelques raisons. . .

1. Le développeur peut faire une erreur dans son code.
2. Le développeur peut télécharger un élément malveillant qui n'aura pas été validé auparavant.
3. Ce qui est échangé sur le réseau peut être chiffré et donc invisible.
4. L'analyste ne peut pas analyser tous les programmes et **doit pouvoir faire confiance** (whitelist).
5. . . .

Conclusion - Cycle de vie SSI d'un logiciel