

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;  
}
```



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.



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.



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.



bonjour_obf.py

```
#!/bin/python3  
  
import base64  
exec(base64.b64decode("cHJpbnQoIkh1bGxvIFdvcmxkICEiKQ=="))
```

Réalise la même chose que le précédent programme.



bonjour.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.



bonjour.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`.



bonjour.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

[illegible]

Exécutable

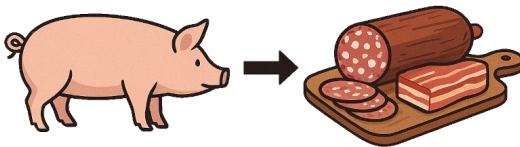
[illegible]

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

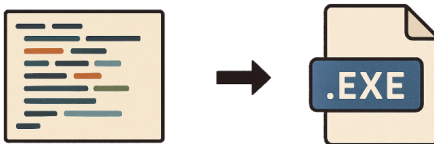


Du code à l'EXE

**DU COCHON
À LA CHARCUTERIE**



**DU CODE
À L'EXE**



Désassembler le programme "bonjour"

The screenshot displays the IDA Pro disassembler interface. The left sidebar shows the 'Functions' list with the following entries: `_init_proc`, `sub_1020`, `sub_1030`, `__cxa_finalize` (highlighted), `_puts`, `_start`, `deregister_tm_clones`, `register_tm_clones`, `_do_global_ctors_aux`, `frame_dummy`, `main`, `_term_proc`, `__libc_start_main`, `_puts`, `__imp__cxa_finalize`, and `__imp__cxa_finalize`. The main window shows the assembly code for the `main` function, which is a 32-bit Intel x86 assembly. The code starts with a comment `; Attributes: bp-based frame` and a `public main` declaration. The function `main` is defined as `main proc near` and `__unwind {`. The code includes a `push ebp` instruction, followed by `mov ebp, esp` and `lea eax, [ebp+4] ; "Bonjour"`. The `call _puts` instruction is used to print the string "Bonjour". The function ends with `ret` and `main endp`. The status bar at the bottom indicates the current address is `00001149` and the function is `main`.

```
; Attributes: bp-based frame
; int __fastcall main(int argc, const char **argv, const char **envp)
public main
main proc near
; __unwind {
endbr64
push ebp
mov  ebp, esp
lea  eax, [ebp+4] ; "Bonjour"
mov  ecx, eax
call _puts
pop  ebp
ret
; } // starts at 1149
main endp
_text ends
```



Assembleur

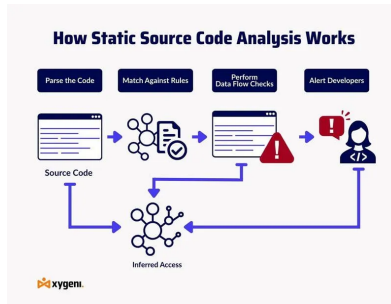
Générer le code "Assembleur" à partir du source : `gcc -S`.

```
.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
```

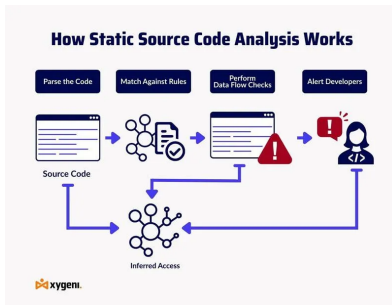
```
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu 13.3.0-6ubuntu2-24.04) 13.3.0"
.section .note.GNU-stack,"",@progbits
.section .note.gnu.property,"a"
.align 8
.long 1f - 0f
.long 4f - 1f
.long 5
0:
.string "GNU"
1:
.align 8
.long 0xc0000002
.long 3f - 2f
2:
.long 0x3
3:
.align 8
4:
```



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...

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



SSI - Quelques raisons...

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



SSI - Quelques raisons...

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



Conclusion - Cycle de vie SSI d'un logiciel

Secure Software Development Life Cycle (SSDLC)

