CYBERSEC 2025

從編譯器視角看 App 程式碼安全與防護

陳忠義

艾斯冰殼股份有限公司

yeecy@iceshell.co

ICE SHELL

About Us

講者

陳忠義

艾斯冰殼 ICEshell
資安編譯器工程師

◆ 專長

- 編譯器最佳化
- 程式碼保護技術研發

共同作者

王羿廷

艾斯冰殼 ICEshell
創辦人暨執行長

◆ 專長

- App 逆向工程
- App 安全防護技術研發

ICE SHELL

Table of Contents

0x0　程式碼保護方法

0x1　現代編譯器架構

0x2　在 LLVM IR 上進行混淆

0x3　如何使用開源方案

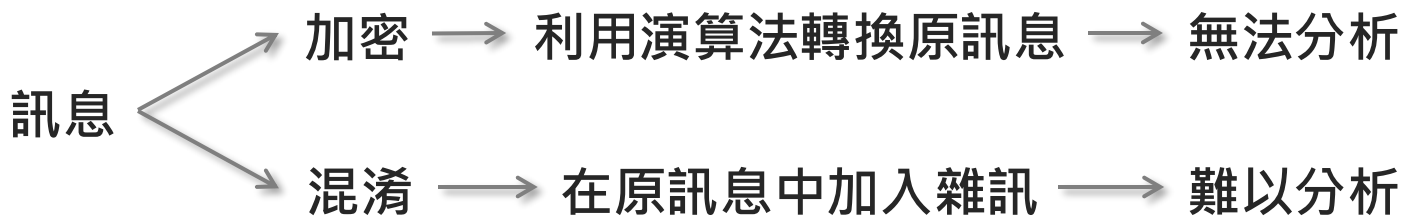
0x4　結語

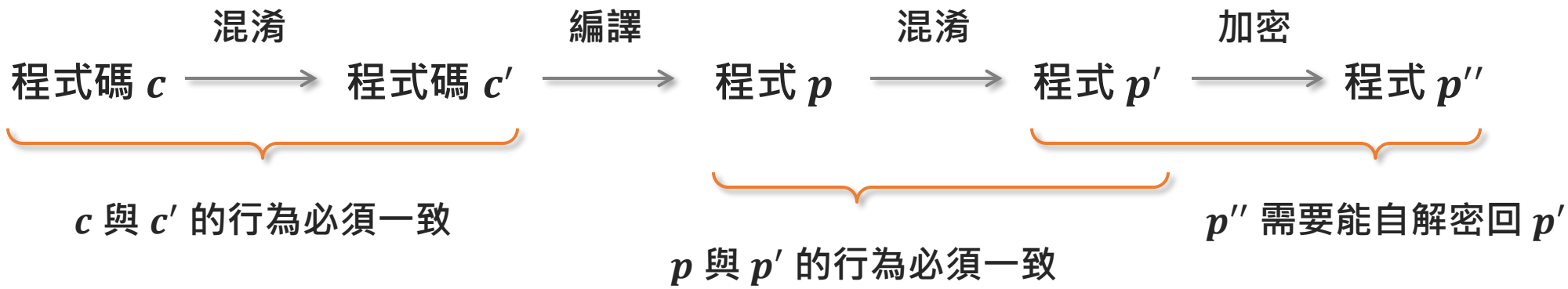
ICE SHELL

程式碼保護方法

ICE SHELL

加密（Encryption）與混淆（Obfuscation）





參考資料：https://medium.com/iceshell-co

現代編譯器架構

ICE SHELL

「編譯器」是什麼？

把某種程式語言翻譯成某種程式語言的程式

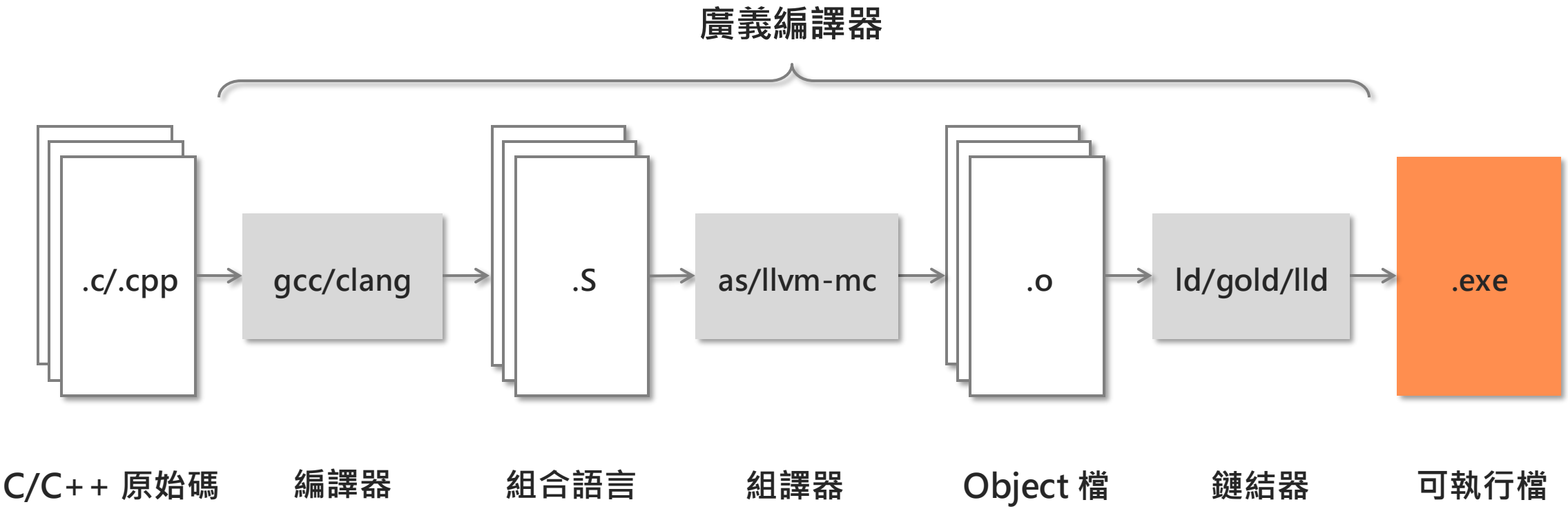
把某個高階程式語言翻譯成某個低階組合語言的程式

1. Google 翻譯不是編譯器 → 看不懂程式語言
2. LLM 不是編譯器 → 翻譯結果不一定正確

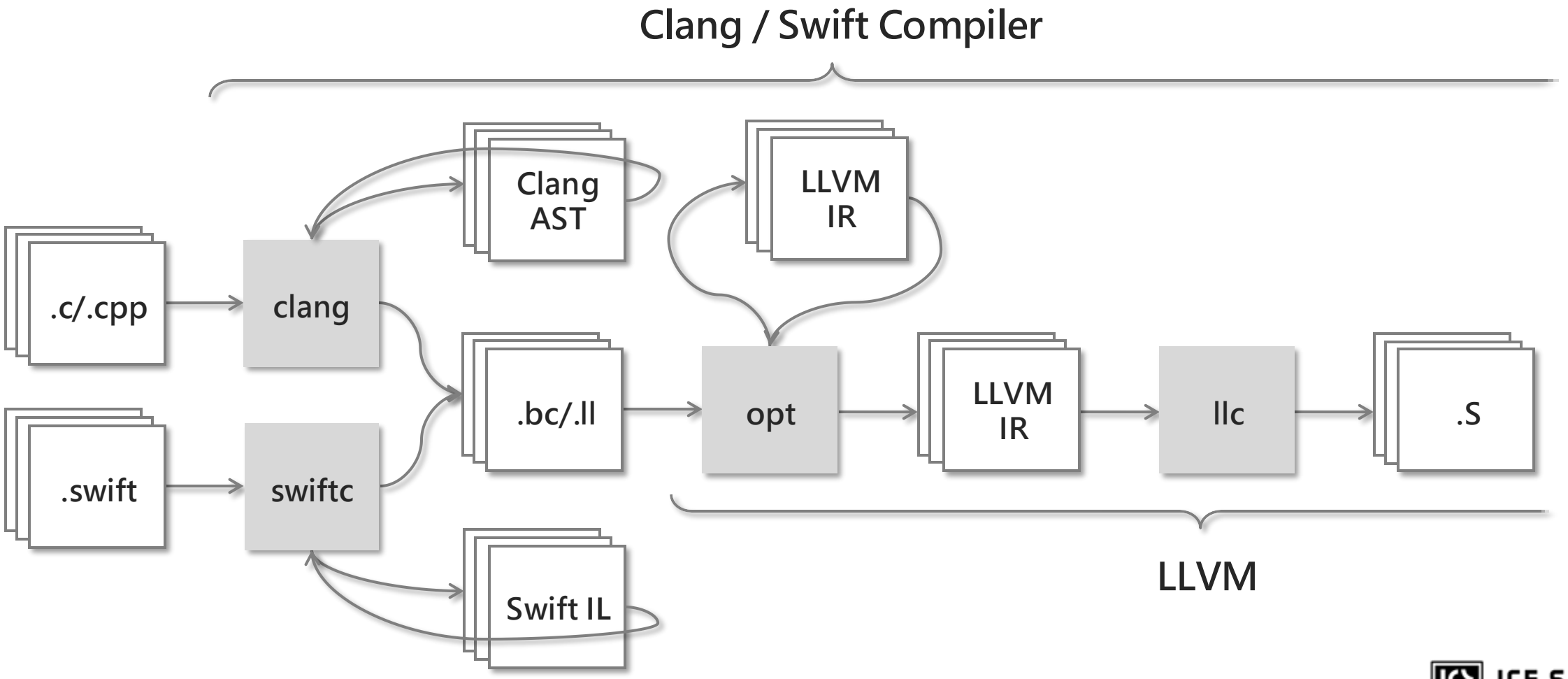
翻譯前後的行為必須一致

如果可以的話，讓翻譯後的程式 (1) 快一點、(2) 小一點或 (3) 亂一點

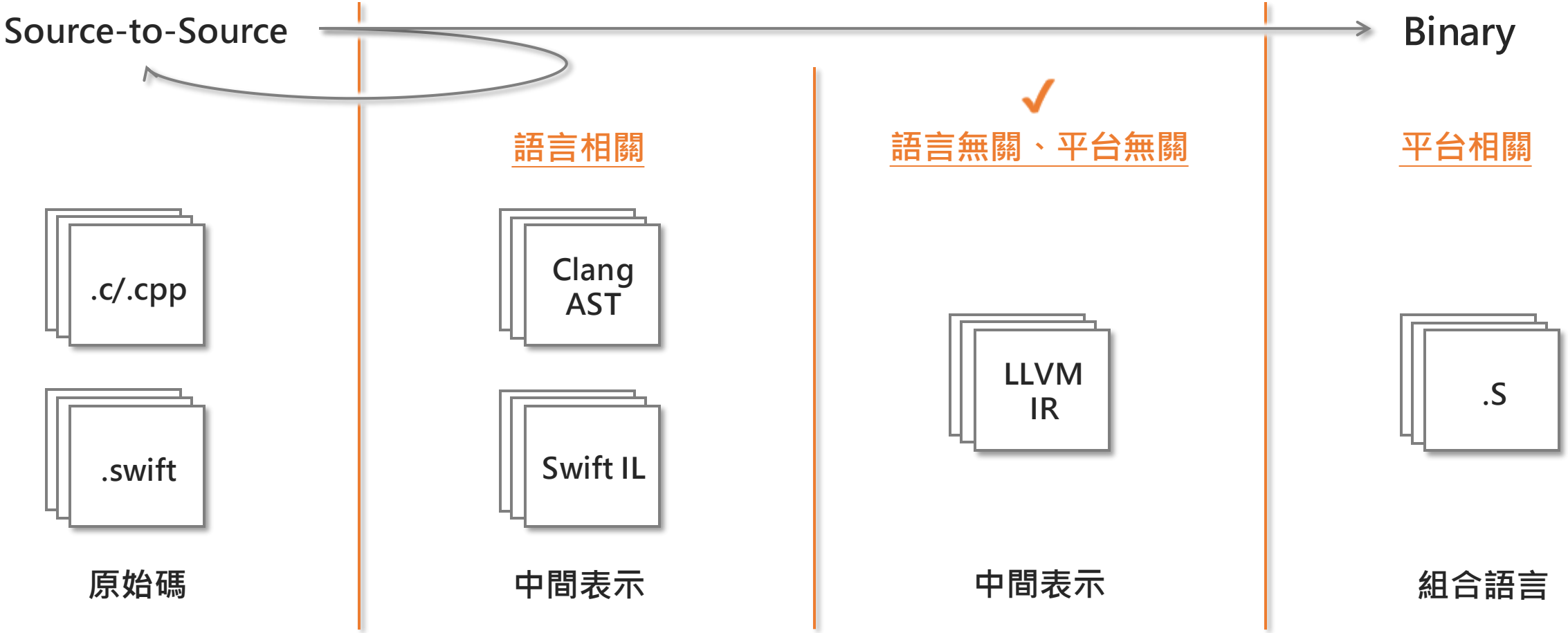
編譯流程：從程式碼到可執行檔



LLVM、Clang 與 Swift Compiler



在哪裡進行混淆？

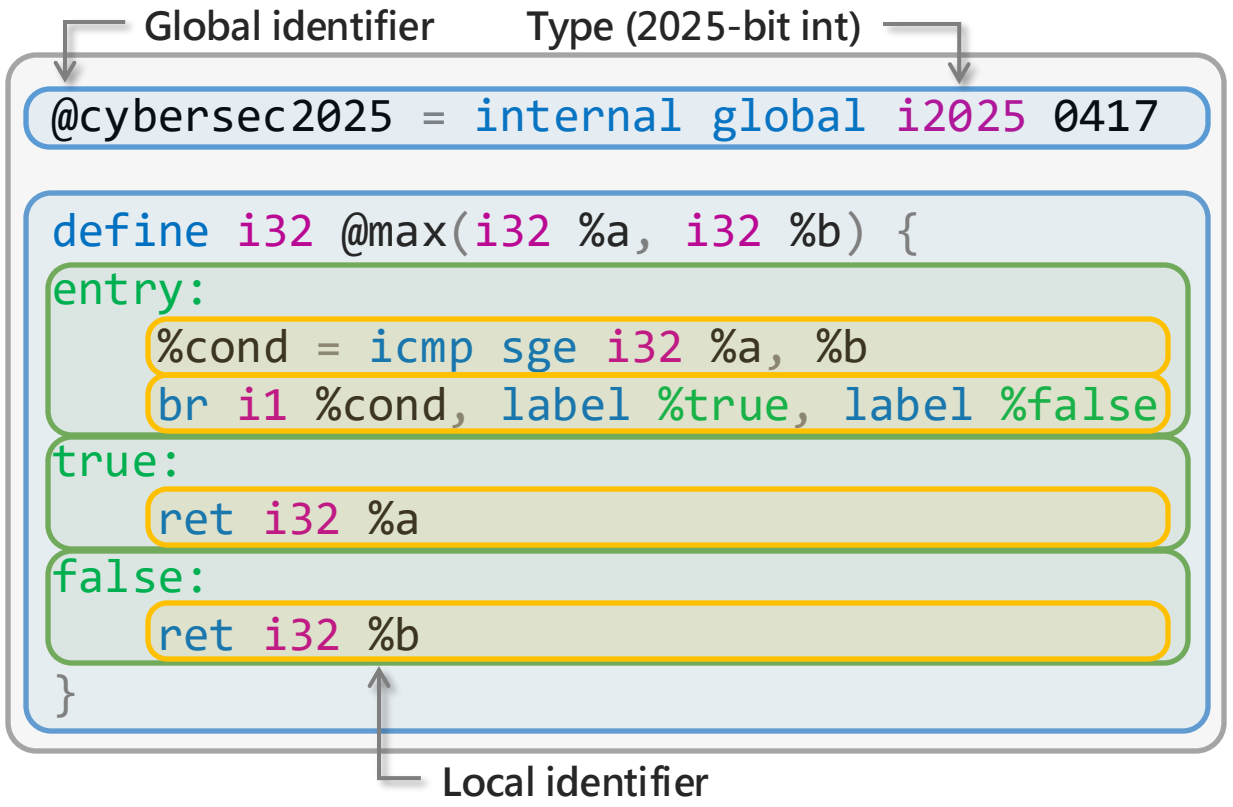


在 LLVM IR 進行混淆

（不是 LLM 喔）

ICE SHELL

LLVM Intermediate Representation



- 低階語言
- 靜態、顯式型別
- 變數不可重複定義

- Module
- GV & Function
- Basic Block
- Instruction

Pipeline & Pass

Pipeline → 一系列的 pass

Pass → 對整個 IR 做某些事的程式碼
- 不會修改 IR → Analysis
- 會修改 IR → Transform

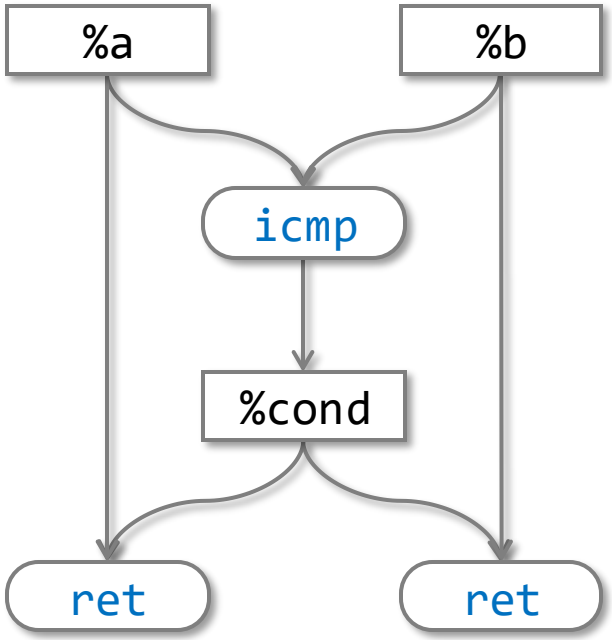
```
@cybersec2025 = internal global i2025 0417

define i32 @max(i32 %a, i32 %b) {
...
}
```

1. 分析變數的使用情況 (analysis)
2. 移除未被使用的變數 (transform)

Data Flow Graph

LLVM IR → DFG

```
@cybersec2025 = internal global i2025 0417

define i32 @max(i32 %a, i32 %b) {
entry:
    %cond = icmp sge i32 %a, %b
    br i1 %cond, label %true, label %false
true:
    ret i32 %a
false:
    ret i32 %b
}
```



DFG 混淆：指令替換

目標：隱藏某些重要的計算模式

代價：程式碼大小 ↑、程式執行時間 ↑

```
%x = xor i32 %a, %b
```

```
a ^ b
= (a & !b) | (!a & b)
```

```
a & b
= (a ^ !b) & a
```

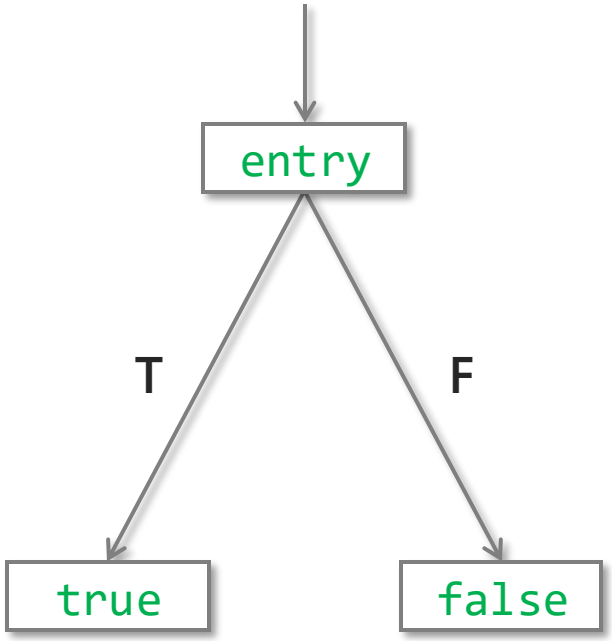
```
  t0 = !a   %t0 = xor i32 %a, 1
  t1 = !b   %t1 = xor i32 %b, 1
t2 = a & t1 %t2 = and i32 %a, %t1
t3 = b & t0 %t3 = and i32 %b, %t0
x = t2 | t3 %x  = or  i32 %t2, %t3
```

```
%t0 = xor i32 %a, 1
%t1 = xor i32 %b, 1
%t4 = xor i32 %t1, 1
%t5 = xor i32 %a, %t4
%t2 = and i32 %t5, %a
%t6 = xor i32 %t0, 1
%t7 = xor i32 %b, %t6
%t3 = and i32 %t7, %b
%x  = or  i32 %t2, %t3
```

Control Flow Graph

LLVM IR → CFG

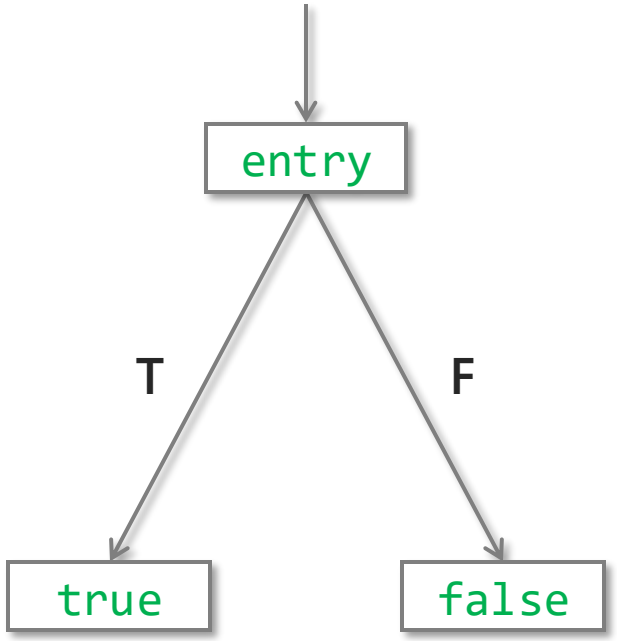
```
@cybersec2025 = internal global i2025 0417

define i32 @max(i32 %a, i32 %b) {
entry:
    %cond = icmp sge i32 %a, %b
    br i1 %cond, label %true, label %false
true:
    ret i32 %a
false:
    ret i32 %b
}
```

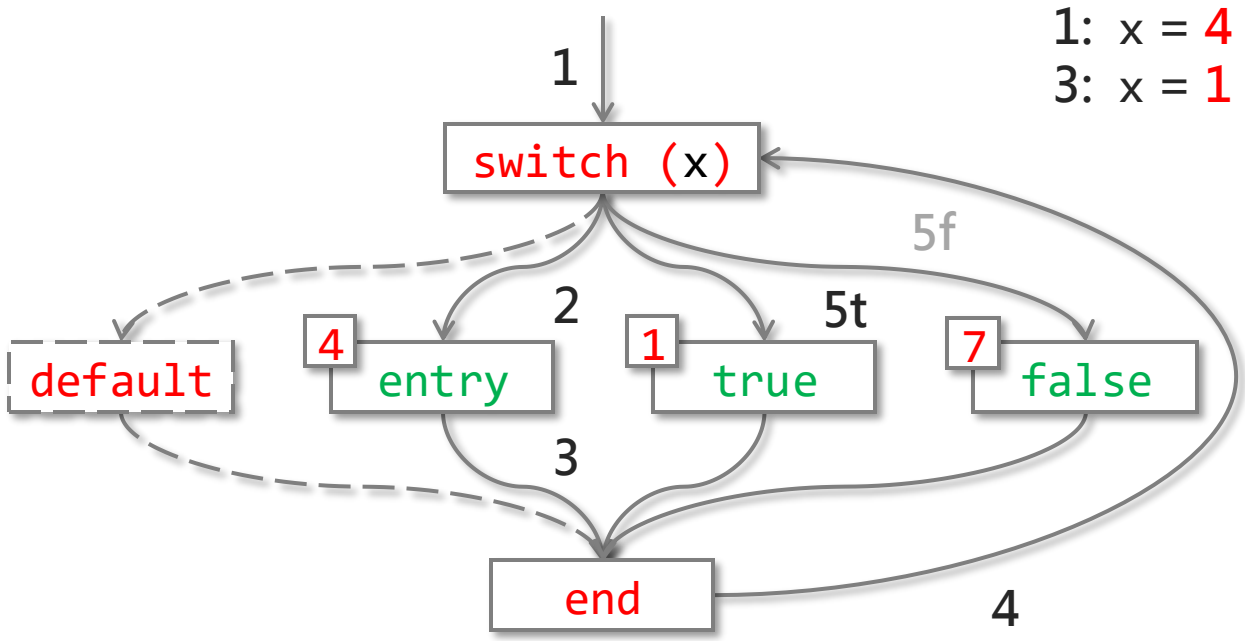


CFG 混淆：Flattening

目標：把控制依賴轉換成資料依賴

代價：程式碼大小 ↑、程式執行時間 ↑ ↑ ↑

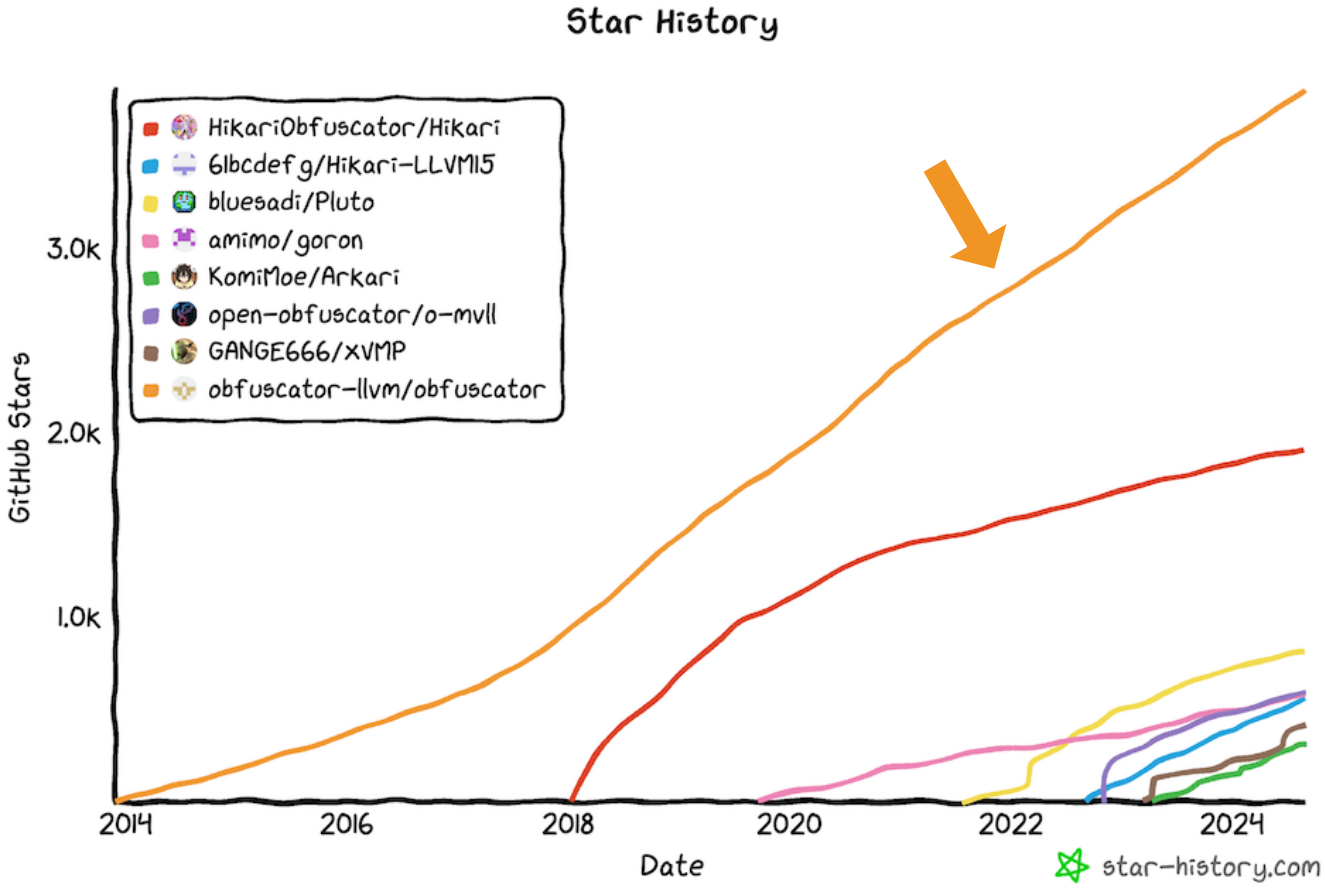




如何使用開源方案

ICE SHELL

Obfuscator-LLVM (OLLVM)



github.com/obfuscator-llvm/obfuscator

- 基於 LLVM 的混淆工具
- 提供三種混淆功能
- 孕育許多相關開源專案
- 2017 年起停止維護

macOS 編譯 Obfuscator-LLVM (1)

```
git clone -b llvm-4.0 https://github.com/obfuscator-llvm/obfuscator.git      && \
cd obfuscator                                                                && \
sed -i '' '20i\
#include "llvm/ADT/StringRef.h"'$'\n' include/llvm/Support/Regex.h           && \
sed -i '' '43i\
#elif defined(__aarch64__)'$'\n' include/llvm/CryptoUtils.h                  && \
sed -i '' '44i\
#ifndef ENDIAN_LITTLE'$'\n'      include/llvm/CryptoUtils.h                  && \
sed -i '' '45i\
#define ENDIAN_LITTLE'$'\n'      include/llvm/CryptoUtils.h                  && \
sed -i '' '46i\
#endif'$'\n'$'\n'                include/llvm/CryptoUtils.h                  && \
```

1 下載原始碼

2 幫忙 include 漏掉的 header

3 讓 OLLVM 支援 AArch64

macOS 編譯 Obfuscator-LLVM (2)

```
sed -i '' '6274 s/&CGF, //' tools/clang/lib/CodeGen/CGOpenMPRuntime.cpp      && \
sed -i '' '6321 s/&CGF, //' tools/clang/lib/CodeGen/CGOpenMPRuntime.cpp      && \
sed -i '' '6400 s/&CGF, //' tools/clang/lib/CodeGen/CGOpenMPRuntime.cpp      && \
mkdir build                                                                  && \
cd build                                                                     && \
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=MinSizeRel                         \
    -DLLVM_TARGETS_TO_BUILD=host ../                                         && \
make -j$(sysctl -n hw.ncpu)
```

4 修正語法問題

5 產生 makefile

6 編譯 OLLVM

- 編譯環境為 Apple M2 + macOS Sonoma 14.4.1 + Apple Clang 15.0.0.15000309
- 不同編譯器會遇到不同編譯警告和錯誤

Linux 編譯 Obfuscator-LLVM

```
git clone -b llvm-4.0 https://github.com/obfuscator-llvm/obfuscator.git                 && \
cd obfuscator                                                                           && \
sed -i '690 s/char/unsigned char/' include/llvm/ExecutionEngine/Orc/OrcRemoteTargetClient.h  && \
sed -i '23irequired_libraries = TransformUtils' lib/Transforms/Obfuscation/LLVMBuild.txt    && \
sed -i '6274 s/&CGF, //' tools/clang/lib/CodeGen/CGOpenMPRuntime.cpp                    && \
sed -i '6321 s/&CGF, //' tools/clang/lib/CodeGen/CGOpenMPRuntime.cpp                    && \
sed -i '6400 s/&CGF, //' tools/clang/lib/CodeGen/CGOpenMPRuntime.cpp                    && \
mkdir build                                                                             && \
cd build                                                                                && \
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=MinSizeRel                                    \
    -DLLVM_TARGETS_TO_BUILD=host -DLLVM_INCLUDE_TESTS=OFF  ../                          && \
make -j$(nproc)
```

- 編譯環境（一）為 Intel Core i5-11400 + Linux 6.8.0-51-generic + GCC 11.4.0
- 編譯環境（二）為 Intel Core i7-1260P + Linux 5.15.167.4-microsoft-standard-WSL2 + GCC 11.4.0
- GNU sed 語法跟 macOS 的 BSD sed 有些微不同

使用 Obfuscator-LLVM 進行混淆

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    puts("CYBERSEC 2025");
    return 0;
}
```

test.c

❗Clang 4.0.1 不認得 Apple Silicon 的 macOS 😭

生成 LLVM IR 再交由原生 Clang 完成後續步驟

```
<ollvm_root>/build/bin/clang -isysroot `xcrun --show-sdk-path` -arch arm64     \
    -S -emit-llvm -mllvm -sub -mllvm -bcf -mllvm -fla -o test.ll test.c     && \
sed -i '' 's/\(target triple = \).*/\1"arm64-apple-macosx14.0.0"/' test.ll  && \
clang -o test test.ll
```

macOS

```
<ollvm_root>/build/bin/clang -mllvm -sub -mllvm -bcf -mllvm -fla -o test test.c
```

Linux

混淆效果

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    puts("CYBERSEC 2025");
    return 0;
}
```

test.c

未混淆反編譯

混淆後反編譯

```
undefined4 entry(void)

{
  _puts("CYBERSEC 2025");
  return 0;
}
```

```
...
  while( true ) {
    while( true ) {
      while (local_30 = local_18, local_18 == -0x7840900a) {
        local_18 = local_20;
        if ((byte)(local_12 & 1 ^ local_11 & 1 | ((local_12 ^ 1 |
          local_18 = local_1c;
        }
      }
      if (local_18 != local_20) break;
      puVar1 = puVar3 + -2;
      puVar2 = puVar3 + -4;
      puVar3 = puVar3 + -6;
      *(undefined4 *)puVar1 = 0;
      *(undefined4 *)puVar2 = local_2c;
      *puVar3 = local_28;
      _puts("CYBERSEC 2025");
      bVar4 = (uint)(__x * (__x + -1)) % 2 == 0;
      local_18 = -0x47d28d82;
      if (bVar4 == __y < 10 && (!bVar4 || __y >= 10)) {
        local_18 = local_1c;
      }
    }
...
```

潛在問題

1. OLLVM 和部分衍生專案的混淆 pass 並未經過適當測試，可能出現編譯錯誤和行為錯誤
2. 編譯器最佳化與混淆的取捨

❗可以將該範例複製貼上到 `xor.ll`，執行以下指令觀察化簡結果

```
<ollvm_root>/build/bin/opt -S --passes=instcombine xor.ll
```

之前的範例 → 化簡結果

```
define i32 @xor(i32 %a, i32 %b) {
    %t0 = xor i32 %a, 1
    %t1 = xor i32 %b, 1
    %t4 = xor i32 %t1, 1
    %t5 = xor i32 %a, %t4
    %t2 = and i32 %t5, %a
    %t6 = xor i32 %t0, 1
    %t7 = xor i32 %b, %t6
    %t3 = and i32 %t7, %b
    %x  = or  i32 %t2, %t3
    ret i32 %x
}
```

```
x = t2 | t3
  = ( t5 & a) | ( t7 & b)
  = ((a ^ t4) & a) | ((b ^ t6) & b)
  = ((a ^ ( t1 ^ 1)) & a) | ((b ^ ( t0 ^ 1)) & b)
  = ((a ^ ((b ^ 1) ^ 1)) & a) | ((b ^ ((a ^ 1) ^ 1)) & b)
  = ((a ^ b) & a) | ((b ^ a) & b)
  = (a ^ b) & (a | b)
  = a ^ b
```

展開

化簡

結語

ICE SHELL

結語

在編譯器上進行混淆是個很自然的想法！

Obfuscator-LLVM 是基於 LLVM 的開源混淆方案

個人使用請多多測試，以免程式行為與預期不一致

商業軟體建議尋求商用混淆方案，編譯器的水很深 □

（想養個 compiler team 嗎？相關人才絕大多數都在晶片大廠喔）

Thanks for Listening

Email: {jason, yeecy}@iceshell.co

- https://github.com/iceshell-co/presentation/tree/main/CYBERSEC-2025

ICE SHELL