Informazioni utili da sapere

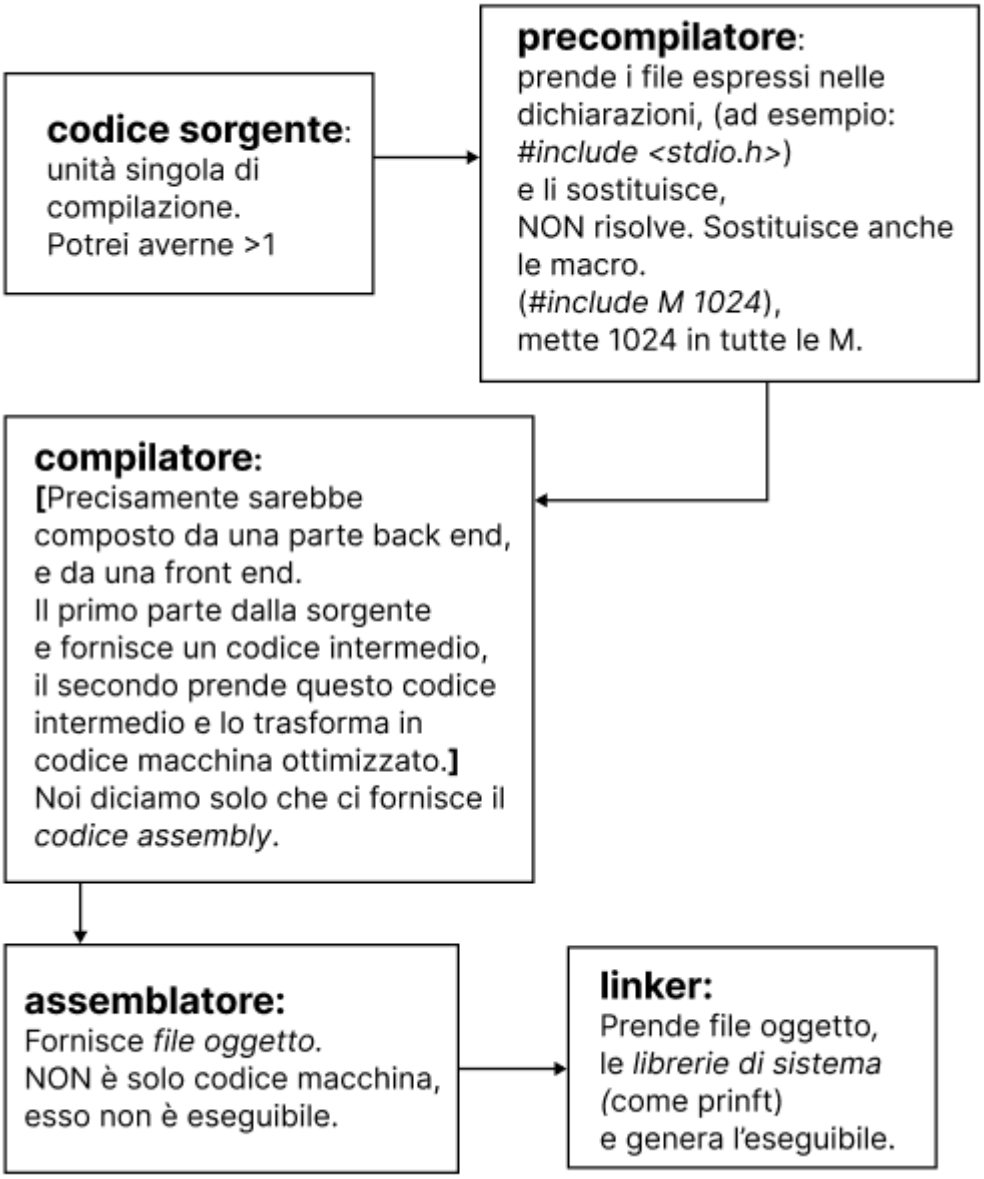
Sistemi operativi consigliati

Poichè la maggior parte dei malware da analizzare sono sviluppati per Windows, ci serve un sistema Windows per testarli. Questo sistema deve però girare su macchina virtuale, non sul computer host, in quanto una gestione non attenta del malware potrebbe portarlo ad operare sul nostro pc di riferimento. Per evitare ciò, sarebbe buona norma avere un sistema operativo diverso da Windows sul nostro pc host.

Virtualizzazione

Per creare macchine virtuali, oltre ai soliti *VmWare* e *VirtualBox*, il professore suggerisce *Qemu*, in quanto è un servizio di paravirtualizzazione efficiente, mentre i primi due sono solo emulazioni software. Qemu ha un apprendimento più ripido, ma propone una efficienza migliore. Non è obbligatorio usare Qemu, in quanto la caratteristica principale che vogliamo sono gli **snapshot**, ovvero la possibilità di generare delle istantanee del sistema. Ad esempio, possiamo fare uno snapshot prima e dopo l'inserimento del malware, e vedere cosa è cambiato.

Ottenere il codice macchina: processo



Flag da GCC

Prendiamo il seguente file che stampa "Hello World"

```c
#include <stdio.h>
int main() {
   // printf() displays the string
   printf("Hello, World!");
   return 0;
}
```

Se compiliamo con `gcc -E test.c|less` otteniamo la sua **precompilazione**, questa è solo una parte, il codice includerebbe anche prototipi, struct, etc...

```
# 0 "test.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "test.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/bits/libc-header-start.h" 1 3 4
# 33 "/usr/include/bits/libc-header-start.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 393 "/usr/include/features.h" 3 4
# 1 "/usr/include/features-time64.h" 1 3 4
# 20 "/usr/include/features-time64.h" 3 4
# 1 "/usr/include/bits/wordsize.h" 1 3 4
# 21 "/usr/include/features-time64.h" 2 3 4
# 1 "/usr/include/bits/timesize.h" 1 3 4
# 19 "/usr/include/bits/timesize.h" 3 4
# 1 "/usr/include/bits/wordsize.h" 1 3 4
# 20 "/usr/include/bits/timesize.h" 2 3 4
# 22 "/usr/include/features-time64.h" 2 3 4
# 394 "/usr/include/features.h" 2 3 4
# 491 "/usr/include/features.h" 3 4
:
```

Se compiliamo con `gcc -S test.c -o test.s` otteniamo il suo **assembler**:

```
        .file   "test.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello, World!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Infine, con `gcc -c test.c -o test.o` otteniamo le **istruzioni macchina**

Possiamo usare `objdump -d test.o` per avere più ordine:

```
test.o:     formato del file elf64-x86-64


Disassemblamento della sezione .text:

0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   bf 00 00 00 00          mov    $0x0,%edi
   9:   b8 00 00 00 00          mov    $0x0,%eax
   e:   e8 00 00 00 00          call   13 <main+0x13>
  13:   b8 00 00 00 00          mov    $0x0,%eax
  18:   5d                      pop    %rbp
  19:   c3                      ret
```

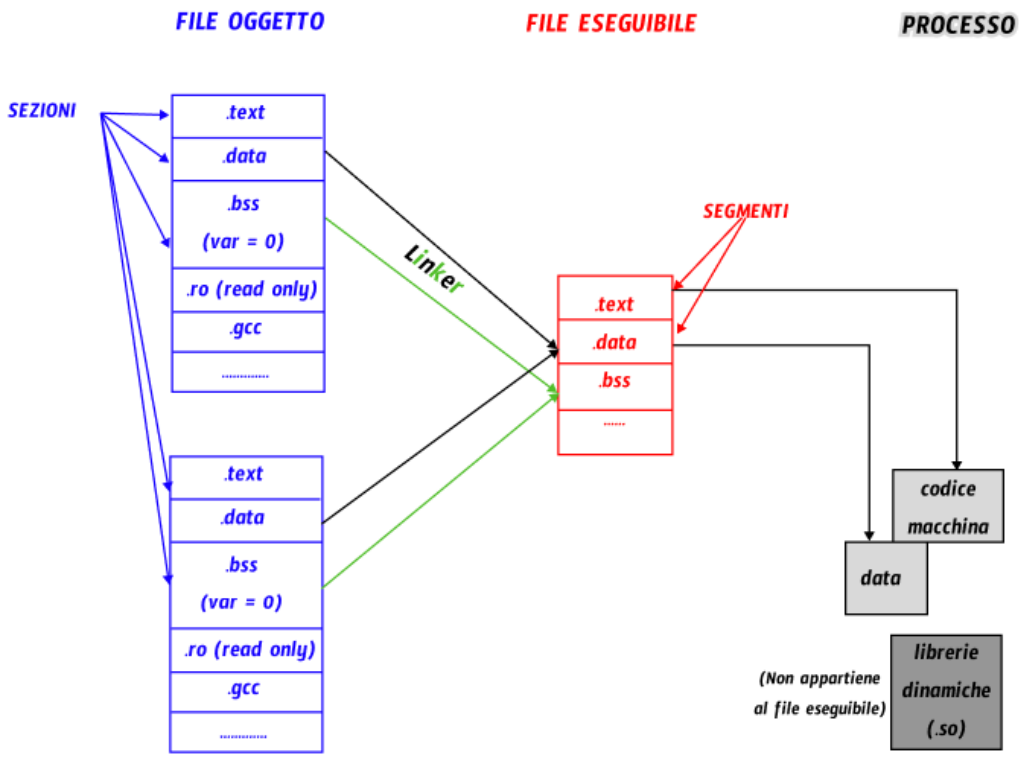
Librerie

- **Statiche**: Se nel file oggetto esiste un riferimento a *printf*, il *linker* mette nel programma eseguibile tutte le istruzioni macchina da *printf*. Vengono incorporate direttamente nell'eseguibile del programma durante la compilazione. Sostituisco le *call prinft* con l'indirizzo della procedura. Il codice è nel file eseguibile, libreria non serve più. Opero a *compile time*.
- **Dinamiche**: Il loro codice non viene incorporato nell'eseguibile, ma avviene un collegamento dinamico a *runtime*. Il linking avviene col sistema operativo, che mantiene queste librerie dinamiche *.dll* o *.so*. Le dimensioni del file sono minori, in quanto nel programma non ho copiato nulla, ma solo collegato.

Esempio: se rimovessi *stdio.h*, ma il mio programma usa le *print*, ho un errore già a livello *assembly*. Questo perchè il *precompilatore* mette il riferimento, per poi delegare il resto al *linker*.

Osservazioni sui file eseguibili

Il file eseguibile NON è un file oggetto. Il file eseguibile contiene tutto il necessario per eseguire. Posso avere anche più file oggetto. Dal *file oggetto* costruisco un *file eseguibile* su *disco* che diventa poi un *processo in RAM*. Un processo è una *istanza di programma in esecuzione, un insieme di informazioni che consentono al programma di tracciare la sua esecuzione.*



I file oggetto (in blu) sono composti da **sezioni**. Il **linker** lega tutte le sezioni di uno stesso tipo *(es: .text)* in un unico **segmento** *(es: .text)*, appartenente al **file eseguibile** (in rosso). Sezioni $\neq$ Segmenti (Possiamo trovare più sezioni rispetto al numero di segmenti.) Passando al **processo**, questo aggiunge elementi non appartenenti al **file eseguibile** (come le librerie dinamiche.) Posso decompilare un file eseguibile, ma non un file oggetto. Questo perchè un *file eseguibile* contiene già il codice macchina tradotto in binari eseguibili, ovvero è più vicino all'hardware. Il *file oggetto*, invece, non è ancora *completo*, poichè dovrà essere combinato con altri file, essendo solo una parte. Inoltre, è possibile incorrere in ottimizzazioni.