UNIVERSITÀ
DEGLI STUDI
DI MILANO

DYNAMIC LINKING

Il dynamic linking è la tecnica utilizzata per eseguire codice esterno al nostro programma di base.
Per esempio, le funzioni printf, scanf, etc fanno parte della libc

```
oot@ubuntu:~/materiale-pwn/dynamic_linking# ldd hello_world
        linux-vdso.so.1 (0x00007ffeeb9b0000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x000077e1e3200000)
        /lib64/ld-linux-x86-64.so.2 (0x000077e1e358b000)
```

Domanda: come fa il programma a sapere dove andare quando viene chiamata una funzione esterna? Osserviamo il disassemblato del programma hello_world

```
0x0000000000001149 <+0>:     endbr64
0x000000000000114d <+4>:     push   rbp
0x000000000000114e <+5>:     mov    rbp,rsp
0x0000000000001151 <+8>:     lea    rax,[rip+0xeac]
0x0000000000001158 <+15>:    mov    rdi,rax
0x000000000000115b <+18>:    call   0x1050 <puts@plt>
0x0000000000001160 <+23>:    mov    eax,0x0
0x0000000000001165 <+28>:    pop    rbp
0x0000000000001166 <+29>:    ret
```

```
Dump of assembler code for function puts@plt:
   0x0000000000001050 <+0>:     endbr64
   0x0000000000001054 <+4>:     jmp    QWORD PTR [rip+0x2f76]
   0x000000000000105a <+10>:    nop    WORD PTR [rax+rax*1+0x0]
End of assembler dump.
```

```
pwndbg> x/1gx 0x3fd0
0x3fd0 <puts@got.plt>:  0x0000000000001030
```

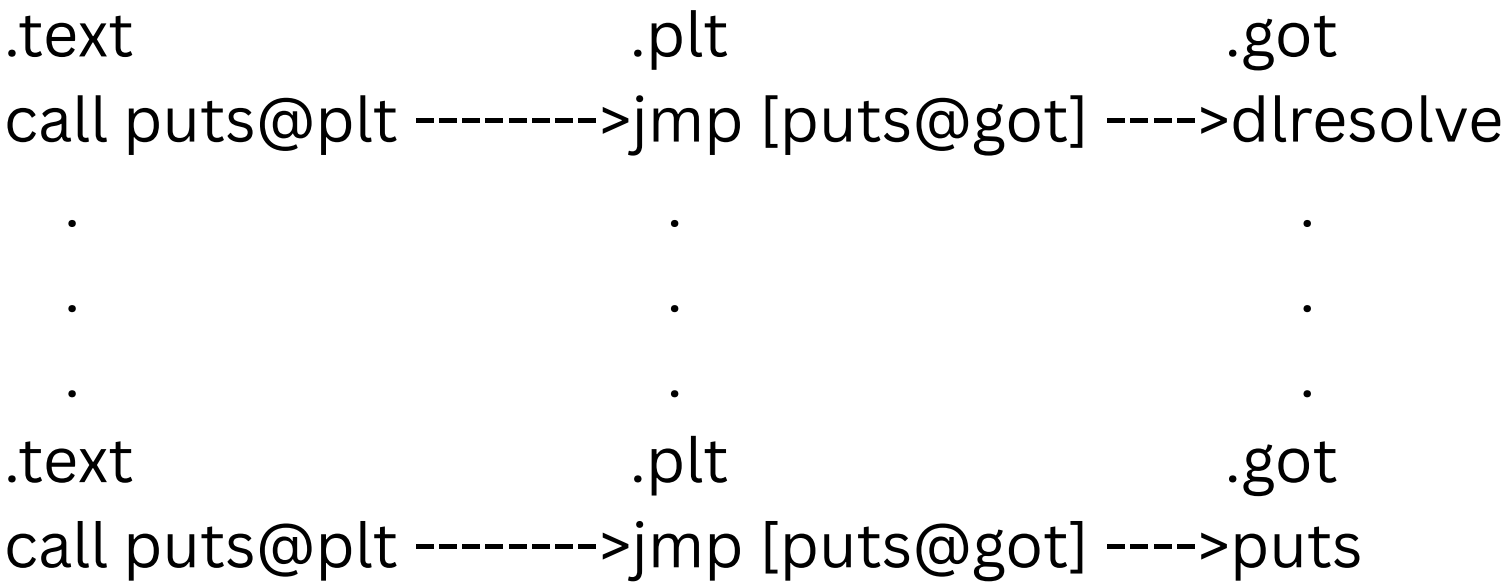
Vediamo che viene preso un valore dalla memoria, e poi il programma salta verso questo...
Ma cosa succede a questo punto della memoria una volta che termina l'esecuzione della funzione?

```
pwndbg> disass
Dump of assembler code for function main:
   0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   rbp
   0x000055555555514e <+5>:     mov    rbp,rsp
   0x0000555555555151 <+8>:     lea    rax,[rip+0xeac]        # 0
   0x0000555555555158 <+15>:    mov    rdi,rax
   0x000055555555515b <+18>:    call   0x555555555050 <puts@plt>
=> 0x0000555555555160 <+23>:    mov    eax,0x0
   0x0000555555555165 <+28>:    pop    rbp
   0x0000555555555166 <+29>:    ret
End of assembler dump.
pwndbg> x/1gx 0x555555557fd0
0x555555557fd0 <puts@got.plt>:  0x00007ffff7c87be0
pwndbg> x/1i 0x00007ffff7c87be0
   0x7ffff7c87be0 <__GI__IO_puts>:      endbr64
```

Quindi Quello che succede è:

- Chiamo la funzione, entrando nella PLT (Procedure Linkage Table)
- Alla prima esecuzione della funzione viene popolata una entry della GOT (Global Offset Table)
- Dalla seconda esecuzione in poi, la GOT entry corrispondente alla funzione contiene l'indirizzo effettivo della funzione che voglio eseguire.

Nota: Questo procedimento viene eseguito per ogni funzione.


.text
.plt
.got
call puts@plt -------->jmp [puts@got] ---->dlresolve
.text
.plt
.got
call puts@plt -------->jmp [puts@got] ---->puts