

## ANALISI di un FILE ELF

Per seguire questa breve analisi, occorre fare riferimento alle definizioni riportate nella pagina “standard ELF” del sito [www.parolamia.eu](http://www.parolamia.eu) oppure scaricare il file in formato PDF. Per iniziare scriviamo e compiliamo un programma di prova.

```
#include <stdio.h>

int var_globale_1 = 3;
int var_globale_2;

int funzione_vuota(void) {
    printf("buongiorno\n");
    return 0;
}

int main(){
int var_locale = 6;
funzione_vuota();
printf("var_globale_1 = %d\n", var_globale_1);
printf("var_globale_2 = %d\n", var_globale_2);
printf("var_locale = %d\n", var_locale);
return 0;
}
```

```
$ gcc -o prova prova.c
```

### HEADER ELF

Usando il programma readelf, contenuto nel pacchetto binutils, possiamo visualizzare l'header ELF

```
$ readelf -h prova
```

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                               ELF32
  Data:                                   2's complement, little endian
  Version:                               1 (current)
  OS/ABI:                                UNIX - System V
  ABI Version:                           0
  Type:                                   EXEC (Executable file)
  Machine:                                Intel 80386
  Version:                                0x1
  Entry point address:                    0x8048310
  Start of program headers:                52 (bytes into file)
  Start of section headers:                2856 (bytes into file)
  Flags:                                   0x0
  Size of this header:                     52 (bytes)
  Size of program headers:                 32 (bytes)
  Number of program headers:                8
  Size of section headers:                 40 (bytes)
  Number of section headers:                35
  Section header string table index:       32
```

Una visione dell'header in esadecimale si ottiene con

**\$ hexdump -C prova | head -n 6** (head filtra le prime 6 righe dell'output di hexdump)

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF.....|
00000010  02 00 03 00 01 00 00 00  10 83 04 08 34 00 00 00  |.....4...|
00000020  28 0b 00 00 00 00 00 00  34 00 20 00 08 00 28 00  |(.....4. ...(.|
00000030  23 00 20 00 06 00 00 00  34 00 00 00 34 80 04 08  |#. ....4...4...|
00000040  34 80 04 08 00 01 00 00  00 01 00 00 05 00 00 00  |4.....|
00000050  04 00 00 00 03 00 00 00  34 01 00 00 34 81 04 08  |.....4...4...|
```

Ritroviamo gli stessi dati, ovviamente coi numeri in formato little-endian, per cui 0x8048310 diventa 10 83 04 08.

## SEZIONI

Le **sezioni** sono le parti del file che contengono le informazioni per il linking; si possono visualizzare con il comando

**\$ readelf -S prova**

There are 35 section headers, starting at offset 0xb28:

Section Headers:

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[ 0]		NULL	00000000	000000	000000	00		0	0	0
[ 1]	.interp	PROGBITS	08048134	000134	000013	00	A	0	0	1
[ 2]	.note.ABI-tag	NOTE	08048148	000148	000020	00	A	0	0	4
[ 3]	.gnu.hash	GNU_HASH	08048168	000168	000020	04	A	4	0	4
[ 4]	.dynsym	DYNSYM	08048188	000188	000060	10	A	5	1	4
[ 5]	.dynstr	STRTAB	080481e8	0001e8	000051	00	A	0	0	1
[ 6]	.gnu.version	VERSYM	0804823a	00023a	00000c	02	A	4	0	2
[ 7]	.gnu.version_r	VERNEED	08048248	000248	000020	00	A	5	1	4
[ 8]	.rel.dyn	REL	08048268	000268	000008	08	A	4	0	4
[ 9]	.rel.plt	REL	08048270	000270	000020	08	A	4	11	4
[10]	.init	PROGBITS	08048290	000290	000030	00	AX	0	0	4
[11]	.plt	PROGBITS	080482c0	0002c0	000050	04	AX	0	0	4
[12]	.text	PROGBITS	08048310	000310	0001cc	00	AX	0	0	16
[13]	.fini	PROGBITS	080484dc	0004dc	00001c	00	AX	0	0	4
[14]	.rodata	PROGBITS	080484f8	0004f8	00004c	00	A	0	0	4
[15]	.eh_frame_hdr	PROGBITS	08048544	000544	00001c	00	A	0	0	4
[16]	.eh_frame	PROGBITS	08048560	000560	000058	00	A	0	0	4
[17]	.ctors	PROGBITS	080495b8	0005b8	000008	00	WA	0	0	4
[18]	.dtors	PROGBITS	080495c0	0005c0	000008	00	WA	0	0	4
[19]	.jcr	PROGBITS	080495c8	0005c8	000004	00	WA	0	0	4
[20]	.dynamic	DYNAMIC	080495cc	0005cc	0000c8	08	WA	5	0	4
[21]	.got	PROGBITS	08049694	000694	000004	04	WA	0	0	4
[22]	.got.plt	PROGBITS	08049698	000698	00001c	04	WA	0	0	4
[23]	.data	PROGBITS	080496b4	0006b4	00000c	00	WA	0	0	4
[24]	.bss	NOBITS	080496c0	0006c0	00000c	00	WA	0	0	4
[25]	.comment	PROGBITS	00000000	0006c0	00006c	00		0	0	1
[26]	.debug_aranges	PROGBITS	00000000	000730	000020	00		0	0	8
[27]	.debug_pubnames	PROGBITS	00000000	000750	000025	00		0	0	1
[28]	.debug_info	PROGBITS	00000000	000775	0000fe	00		0	0	1
[29]	.debug_abbrev	PROGBITS	00000000	000873	00005f	00		0	0	1
[30]	.debug_line	PROGBITS	00000000	0008d2	000082	00		0	0	1
[31]	.debug_str	PROGBITS	00000000	000954	00009a	01	MS	0	0	1
[32]	.shstrtab	STRTAB	00000000	0009ee	000139	00		0	0	1
[33]	.symtab	SYMTAB	00000000	0010a0	0004d0	10		34	53	4
[34]	.strtab	STRTAB	00000000	001570	00024a	00		0	0	1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)

I (info), L (link order), G (group), x (unknown)  
0 (extra OS processing required) o (OS specific), p (processor specific)

Il contenuto esadecimale delle sezioni si può visualizzare con

### \$ objdump -s prova

prova: file format elf32-i386

```
Contents of section .interp:
8048134 2f6c6962 2f6c642d 6c696e75 782e736f /lib/ld-linux.so
8048144 2e3200 .2.
Contents of section .note.ABI-tag:
8048148 04000000 10000000 01000000 474e5500 .....GNU.
8048158 00000000 02000000 06000000 09000000 .....
Contents of section .gnu.hash:
8048168 02000000 05000000 01000000 05000000 .....
8048178 00200020 00000000 05000000 ad4be3c0 . . . . .K..
Contents of section .dynsym:
8048188 00000000 00000000 00000000 00000000 .....
8048198 01000000 00000000 00000000 20000000 .....
80481a8 35000000 00000000 00000000 12000000 5.....
80481b8 2e000000 00000000 00000000 12000000 .....
80481c8 29000000 00000000 00000000 12000000 ).....
80481d8 1a000000 fc840408 04000000 11000e00 .....
Contents of section .dynstr:
80481e8 005f5f67 6d6f6e5f 73746172 745f5f00 .__gmon_start__.
80481f8 6c696263 2e736f2e 36005f49 4f5f7374 libc.so.6._IO_st
8048208 64696e5f 75736564 00707574 73007072 din_used.puts.pr
8048218 696e7466 005f5f6c 6962635f 73746172 intf.__libc_star
8048228 745f6d61 696e0047 4c494243 5f322e30 t_main.GLIBC_2.0
8048238 00 .
Contents of section .gnu.version:
804823a 00000000 02000200 02000100 .....
Contents of section .gnu.version_r:
8048248 01000100 10000000 10000000 00000000 .....
8048258 1069690d 00000200 47000000 00000000 .ii.....G.....
Contents of section .rel.dyn:
8048268 94960408 06010000 .....
Contents of section .rel.plt:
8048270 a4960408 07010000 a8960408 07020000 .....
8048280 ac960408 07030000 b0960408 07040000 .....
Contents of section .init:
8048290 5589e553 83ec04e8 00000000 5b81c3fc U..S.....[...
80482a0 1300008b 93fcffff ff85d274 05e81e00 .....t....
80482b0 0000e8e9 000000e8 f4010000 585bc9c3 .....X[...
Contents of section .plt:
80482c0 ff359c96 0408ff25 a0960408 00000000 .5.....%.....
80482d0 ff25a496 04086800 000000e9 e0ffffff .%....h.....
80482e0 ff25a896 04086808 000000e9 d0ffffff .%....h.....
80482f0 ff25ac96 04086810 000000e9 c0ffffff .%....h.....
8048300 ff25b096 04086818 000000e9 b0ffffff .%....h.....
Contents of section .text:
8048310 31ed5e89 e183e4f0 50545268 40840408 1.^.....PTRh@...
8048320 68508404 08515668 dd830408 e8afffff hP...QVh.....
8048330 fff49090 90909090 90909090 90909090 .....
8048340 5589e553 83ec0480 3dc09604 0800753f U..S....=.....u?
8048350 a1c49604 08bbc495 040881eb c0950408 .....
8048360 c1fb0283 eb0139d8 731e8db6 00000000 .....9.s.....
8048370 83c001a3 c4960408 ff1485c0 950408a1 .....
```

```

8048380 c4960408 39d872e8 c605c096 04080183 .....9.r.....
8048390 c4045b5d c38d7426 008dbc27 00000000 ..[.]..t&...'....
80483a0 5589e583 ec18a1c8 95040885 c07412b8 U.....t..
80483b0 00000000 85c07409 c70424c8 950408ff .....t...$.
80483c0 d0c9c390 5589e583 ec18c704 24008504 .....U.....$.
80483d0 08e82aff fffffb800 000000c9 c35589e5 ..*.....U..
80483e0 83e4f083 ec20c744 241c0600 0000e8d1 .....D$.
80483f0 fffffff8b 15bc9604 08b80b85 04088954 .....T
8048400 24048904 24e8e6fe ffff8b15 c8960408 $...$.
8048410 b81f8504 08895424 04890424 e8cffe9f .....T$.
8048420 ffb83385 04088b54 241c8954 24048904 ..3....T$.
8048430 24e8baf8 fffffb800 000000c9 c3909090 $.
8048440 5589e55d c38d7426 008dbc27 00000000 U..]..t&...'....
8048450 5589e557 5653e84f 00000081 c33d1200 U..WVS.0.....=.
8048460 0083ec1c e827fe9f ff8dbb20 fffffff8d .....'.
8048470 8320ffff ff29c7c1 ff0285ff 742431f6 . ...).
8048480 8b451089 4424088b 450c8944 24048b45 .E..D$.E..D$.E
8048490 08890424 ff94b320 fffffff83 c60139fe ...$.
80484a0 72de83c4 1c5b5e5f 5dc38b1c 24c39090 r....[^_]...$.
80484b0 5589e553 83ec04a1 b8950408 83f8ff74 U..S.....t
80484c0 13bbb895 04086690 83eb04ff d08b0383 .....f.....
80484d0 f8ff75f4 83c4045b 5dc39090 ..u....[]...
Contents of section .fini:
80484dc 5589e553 83ec04e8 00000000 5b81c3b0 U..S.....[...
80484ec 110000e8 4cfeffff 595bc9c3 ....L...Y[...
Contents of section .rodata:
80484f8 03000000 01000200 62756f6e 67696f72 .....buongior
8048508 6e6f0076 61725f67 6c6f6261 6c655f31 no.var_globale_1
8048518 203d2025 640a0076 61725f67 6c6f6261 = %d..var_globa
8048528 6c655f32 203d2025 640a0076 61725f6c le_2 = %d..var_l
8048538 6f63616c 65203d20 25640a00 .....ocale = %d..
Contents of section .eh_frame_hdr:
8048544 011b033b 18000000 02000000 fcfeffff ...;.
8048554 34000000 0cffffff 50000000 4.....P...
Contents of section .eh_frame:
8048560 14000000 00000000 017a5200 017c0801 .....zR...|..
8048570 1b0c0404 88010000 18000000 1c000000 .....
8048580 c0feffff 05000000 00410e08 4285020d .....A..B...
8048590 05000000 1c000000 38000000 b4feffff .....8.....
80485a0 5a000000 00410e08 4285020d 054e8305 Z....A..B....N..
80485b0 86048703 00000000 .....
Contents of section .ctors:
80495b8 ffffffff 00000000 .....
Contents of section .dtors:
80495c0 ffffffff 00000000 .....
Contents of section .jcr:
80495c8 00000000 .....
Contents of section .dynamic:
80495cc 01000000 10000000 0c000000 90820408 .....
80495dc 0d000000 dc840408 f5feff6f 68810408 .....oh...
80495ec 05000000 e8810408 06000000 88810408 .....
80495fc 0a000000 51000000 0b000000 10000000 ....Q.....
804960c 15000000 00000000 03000000 98960408 .....
804961c 02000000 20000000 14000000 11000000 ....
804962c 17000000 70820408 11000000 68820408 ....p.....h...
804963c 12000000 08000000 13000000 08000000 .....
804964c feffff6f 48820408 fffffff6f 01000000 ...oH.....o...
804965c f0ffff6f 3a820408 00000000 00000000 ...o:.....
804966c 00000000 00000000 00000000 00000000 .....
804967c 00000000 00000000 00000000 00000000 .....
804968c 00000000 00000000 .....
Contents of section .got:
8049694 00000000 .....
Contents of section .got.plt:

```

```

8049698 cc950408 00000000 00000000 d6820408 .....
80496a8 e6820408 f6820408 06830408 .....
Contents of section .data:
80496b4 00000000 00000000 03000000 .....
Contents of section .comment:
0000 00474343 3a202847 4e552920 342e342e .GCC: (GNU) 4.4.
0010 31000047 43433a20 28474e55 2920342e 1..GCC: (GNU) 4.
0020 342e3100 00474343 3a202847 4e552920 4.1..GCC: (GNU)
0030 342e342e 31000047 43433a20 28474e55 4.4.1..GCC: (GNU
0040 2920342e 342e3100 00474343 3a202847 ) 4.4.1..GCC: (G
0050 4e552920 342e342e 31000047 43433a20 NU) 4.4.1..GCC:
0060 28474e55 2920342e 342e3100 (GNU) 4.4.1.
Contents of section .debug_aranges:
0000 1c000000 02000000 00000400 00000000 .....
0010 10830408 22000000 00000000 00000000 ....".....
Contents of section .debug_pubnames:
0000 21000000 02007100 00008d00 00007500 !.....q.....u.
0010 00005f49 4f5f7374 64696e5f 75736564 .._IO_stdin_used
0020 00000000 00 .....
Contents of section .debug_info:
0000 6d000000 02000000 00000401 00000000 m.....
0010 10830408 32830408 2e2e2f73 79736465 ....2...../sysde
0020 70732f69 3338362f 656c662f 73746172 ps/i386/elf/star
0030 742e5300 2f686f6d 652f7161 7465616d t.S./home/qateam
0040 2f72706d 2f425549 4c442f67 6c696263 /rpm/BUILD/glibc
0050 2d322e31 302e312f 63737500 474e5520 -2.10.1/csu.GNU
0060 41532032 2e31392e 35312e30 2e320001 AS 2.19.51.0.2..
0070 80890000 00020014 00000004 01130000 .....
0080 00018500 00005d00 00003483 04083483 .....]...4...4.
0090 04085b00 00000201 084f0000 00020207 ..[.....0.....
00a0 00000000 02040742 00000002 04073d00 .....B.....=.
00b0 00000201 06510000 00020205 1f000000 .....Q.....
00c0 03040569 6e740002 08058c00 00000208 ...int.....
00d0 07380000 00020405 91000000 04040702 .8.....
00e0 01065800 00000529 00000001 19870000 ..X.....).....
00f0 00010503 fc840408 064f0000 0000 .....0....
Contents of section .debug_abbrev:
0000 01110010 06110112 0103081b 08250813 .....%..
0010 05000000 01110125 0e130b03 0e1b0e11 .....%.....
0020 01120110 06000002 24000b0b 3e0b030e .....$...>...
0030 00000324 000b0b3e 0b030800 00042400 ...$.>.....$.
0040 0b0b3e0b 00000534 00030e3a 0b3b0b49 ..>...4...:.;.I
0050 133f0c02 0a000006 26004913 00000000 .?.....&.I....
Contents of section .debug_line:
0000 57000000 02003200 00000101 fb0e0d00 W.....2.....
0010 01010101 00000001 0000012e 2e2f7379 ...../sy
0020 73646570 732f6933 38362f65 6c660000 sdeps/i386/elf..
0030 73746172 742e5300 01000000 00050210 start.S.....
0040 83040803 c0000133 21343d25 22031820 .....3!4=%"..
0050 595a2122 5c5b0201 00010123 00000002 YZ!"\[.....#....
0060 001d0000 000101fb 0e0d0001 01010100 .....
0070 00000100 00010069 6e69742e 63000000 .....init.c...
0080 0000 .....
Contents of section .debug_str:
0000 73686f72 7420756e 7369676e 65642069 short unsigned i
0010 6e740047 4e552043 20342e34 2e310073 nt.GNU C 4.4.1.s
0020 686f7274 20696e74 005f494f 5f737464 hort int._IO_std
0030 696e5f75 73656400 6c6f6e67 206c6f6e in_used.long lon
0040 6720756e 7369676e 65642069 6e740075 g unsigned int.u
0050 6e736967 6e656420 63686172 002f686f nsigned char./ho
0060 6d652f71 61746561 6d2f7270 6d2f4255 me/qateam/rpm/BU
0070 494c442f 676c6962 632d322e 31302e31 ILD/glibc-2.10.1
0080 2f637375 00696e69 742e6300 6c6f6e67 /csu.init.c.long
0090 206c6f6e 6720696e 7400 long int.

```

## SEZIONE .TEXT

La sezione che contiene il codice del programma è .text. Per isolarla si usa il comando seguente; l'opzione A stampa le n righe che seguono quella che contiene il testo cercato.

```
$ objdump -s prova | grep .text -A 29
```

```
Contents of section .text:
```

```
8048310 31ed5e89 e183e4f0 50545268 40840408 1.^.....PTRh@...
8048320 68508404 08515668 dd830408 e8afffff hP...QVh.....
```

```
. . . . .
```

```
80484c0 13bbb895 04086690 83eb04ff d08b0383 .....f.....
80484d0 f8ff75f4 83c4045b 5dc39090 ..u....[]...
```

Si può visualizzare il contenuto disassemblato con il comando seguente; l'opzione M forza la notazione intel.

```
$ objdump -M intel -d -j .text prova
```

```
prova:      file format elf32-i386
```

```
Disassembly of section .text:
```

```
08048310 <_start>:
```

```
8048310:  31 ed                xor    ebp,ebp
8048312:  5e                  pop    esi
8048313:  89 e1              mov    ecx,esp
8048315:  83 e4 f0          and    esp,0xfffffff0
8048318:  50                push  eax
8048319:  54                push  esp
804831a:  52                push  edx
804831b:  68 40 84 04 08    push  0x8048440
8048320:  68 50 84 04 08    push  0x8048450
8048325:  51                push  ecx
8048326:  56                push  esi
8048327:  68 dd 83 04 08    push  0x80483dd
804832c:  e8 af ff ff ff    call  80482e0 <__libc_start_main@plt>
8048331:  f4                hlt
8048332:  90                nop
8048333:  90                nop
8048334:  90                nop
8048335:  90                nop
8048336:  90                nop
8048337:  90                nop
8048338:  90                nop
8048339:  90                nop
804833a:  90                nop
804833b:  90                nop
804833c:  90                nop
804833d:  90                nop
804833e:  90                nop
804833f:  90                nop
```

```
08048340 <__do_global_dtors_aux>:
```

```
8048340:  55                push  ebp
```

```

8048341: 89 e5          mov     ebp, esp
8048343: 53            push   ebx
8048344: 83 ec 04      sub     esp, 0x4
8048347: 80 3d c0 96 04 08 00  cmp    BYTE PTR ds:0x80496c0, 0x0
804834e: 75 3f        jne    804838f <__do_global_dtors_aux+0x4f>
8048350: a1 c4 96 04 08  mov    eax, ds:0x80496c4
8048355: bb c4 95 04 08  mov    ebx, 0x80495c4
804835a: 81 eb c0 95 04 08  sub    ebx, 0x80495c0
8048360: c1 fb 02      sar    ebx, 0x2
8048363: 83 eb 01      sub    ebx, 0x1
8048366: 39 d8        cmp    eax, ebx
8048368: 73 1e        jae    8048388 <__do_global_dtors_aux+0x48>
804836a: 8d b6 00 00 00 00  lea    esi, [esi+0x0]
8048370: 83 c0 01      add    eax, 0x1
8048373: a3 c4 96 04 08  mov    ds:0x80496c4, eax
8048378: ff 14 85 c0 95 04 08  call   DWORD PTR [eax*4+0x80495c0]
804837f: a1 c4 96 04 08  mov    eax, ds:0x80496c4
8048384: 39 d8        cmp    eax, ebx
8048386: 72 e8        jb    8048370 <__do_global_dtors_aux+0x30>
8048388: c6 05 c0 96 04 08 01  mov    BYTE PTR ds:0x80496c0, 0x1
804838f: 83 c4 04      add    esp, 0x4
8048392: 5b          pop    ebx
8048393: 5d          pop    ebp
8048394: c3          ret
8048395: 8d 74 26 00  lea    esi, [esi+eiz*1+0x0]
8048399: 8d bc 27 00 00 00 00  lea    edi, [edi+eiz*1+0x0]

```

080483a0 <frame\_dummy>:

```

80483a0: 55          push   ebp
80483a1: 89 e5      mov    ebp, esp
80483a3: 83 ec 18   sub    esp, 0x18
80483a6: a1 c8 95 04 08  mov    eax, ds:0x80495c8
80483ab: 85 c0     test   eax, eax
80483ad: 74 12     je     80483c1 <frame_dummy+0x21>
80483af: b8 00 00 00 00  mov    eax, 0x0
80483b4: 85 c0     test   eax, eax
80483b6: 74 09     je     80483c1 <frame_dummy+0x21>
80483b8: c7 04 24 c8 95 04 08  mov    DWORD PTR [esp], 0x80495c8
80483bf: ff d0     call  eax
80483c1: c9       leave
80483c2: c3       ret
80483c3: 90       nop

```

080483c4 <funzione\_vuota>:

```

80483c4: 55          push   ebp
80483c5: 89 e5      mov    ebp, esp
80483c7: 83 ec 18   sub    esp, 0x18
80483ca: c7 04 24 00 85 04 08  mov    DWORD PTR [esp], 0x8048500
80483d1: e8 2a ff ff ff  call   8048300 <puts@plt>
80483d6: b8 00 00 00 00  mov    eax, 0x0
80483db: c9       leave
80483dc: c3       ret

```

080483dd <main>:

```

80483dd: 55          push   ebp
80483de: 89 e5      mov    ebp, esp
80483e0: 83 e4 f0   and    esp, 0xffffffff
80483e3: 83 ec 20   sub    esp, 0x20
80483e6: c7 44 24 1c 06 00 00  mov    DWORD PTR [esp+0x1c], 0x6
80483ed: 00
80483ee: e8 d1 ff ff ff  call   80483c4 <funzione_vuota>
80483f3: 8b 15 bc 96 04 08  mov    edx, DWORD PTR ds:0x80496bc
80483f9: b8 0b 85 04 08  mov    eax, 0x804850b
80483fe: 89 54 24 04  mov    DWORD PTR [esp+0x4], edx

```

```

8048402: 89 04 24          mov     DWORD PTR [esp],eax
8048405: e8 e6 fe ff ff   call   80482f0 <printf@plt>
804840a: 8b 15 c8 96 04 08 mov     edx,DWORD PTR ds:0x80496c8
8048410: b8 1f 85 04 08   mov     eax,0x804851f
8048415: 89 54 24 04      mov     DWORD PTR [esp+0x4],edx
8048419: 89 04 24          mov     DWORD PTR [esp],eax
804841c: e8 cf fe ff ff   call   80482f0 <printf@plt>
8048421: b8 33 85 04 08   mov     eax,0x8048533
8048426: 8b 54 24 1c      mov     edx,DWORD PTR [esp+0x1c]
804842a: 89 54 24 04      mov     DWORD PTR [esp+0x4],edx
804842e: 89 04 24          mov     DWORD PTR [esp],eax
8048431: e8 ba fe ff ff   call   80482f0 <printf@plt>
8048436: b8 00 00 00 00   mov     eax,0x0
804843b: c9              leave
804843c: c3              ret
804843d: 90              nop
804843e: 90              nop
804843f: 90              nop

08048440 <__libc_csu_fini>:
8048440: 55              push   ebp
8048441: 89 e5           mov     ebp,esp
8048443: 5d              pop     ebp
8048444: c3              ret
8048445: 8d 74 26 00     lea    esi,[esi+eiz*1+0x0]
8048449: 8d bc 27 00 00 00 lea    edi,[edi+eiz*1+0x0]

08048450 <__libc_csu_init>:
8048450: 55              push   ebp
8048451: 89 e5           mov     ebp,esp
8048453: 57              push   edi
8048454: 56              push   esi
8048455: 53              push   ebx
8048456: e8 4f 00 00 00   call   80484aa <__i686.get_pc_thunk.bx>
804845b: 81 c3 3d 12 00 00 add     ebx,0x123d
8048461: 83 ec 1c        sub     esp,0x1c
8048464: e8 27 fe ff ff   call   8048290 <_init>
8048469: 8d bb 20 ff ff ff lea    edi,[ebx-0xe0]
804846f: 8d 83 20 ff ff ff lea    eax,[ebx-0xe0]
8048475: 29 c7          sub     edi,eax
8048477: c1 ff 02        sar     edi,0x2
804847a: 85 ff          test   edi,edi
804847c: 74 24          je     80484a2 <__libc_csu_init+0x52>
804847e: 31 f6          xor     esi,esi
8048480: 8b 45 10        mov     eax,DWORD PTR [ebp+0x10]
8048483: 89 44 24 08     mov     DWORD PTR [esp+0x8],eax
8048487: 8b 45 0c        mov     eax,DWORD PTR [ebp+0xc]
804848a: 89 44 24 04     mov     DWORD PTR [esp+0x4],eax
804848e: 8b 45 08        mov     eax,DWORD PTR [ebp+0x8]
8048491: 89 04 24        mov     DWORD PTR [esp],eax
8048494: ff 94 b3 20 ff ff ff call   DWORD PTR [ebx+esi*4-0xe0]
804849b: 83 c6 01        add     esi,0x1
804849e: 39 fe          cmp     esi,edi
80484a0: 72 de          jb     8048480 <__libc_csu_init+0x30>
80484a2: 83 c4 1c        add     esp,0x1c
80484a5: 5b              pop     ebx
80484a6: 5e              pop     esi
80484a7: 5f              pop     edi
80484a8: 5d              pop     ebp
80484a9: c3              ret

080484aa <__i686.get_pc_thunk.bx>:
80484aa: 8b 1c 24        mov     ebx,DWORD PTR [esp]
80484ad: c3              ret

```

```

80484ae: 90          nop
80484af: 90          nop

080484b0 <__do_global_ctors_aux>:
80484b0: 55          push    ebp
80484b1: 89 e5      mov     ebp,esp
80484b3: 53          push    ebx
80484b4: 83 ec 04   sub     esp,0x4
80484b7: a1 b8 95 04 08 mov    eax,ds:0x80495b8
80484bc: 83 f8 ff   cmp    eax,0xffffffff
80484bf: 74 13      je     80484d4 <__do_global_ctors_aux+0x24>
80484c1: bb b8 95 04 08 mov    ebx,0x80495b8
80484c6: 66 90      xchg   ax,ax
80484c8: 83 eb 04   sub    ebx,0x4
80484cb: ff d0     call   eax
80484cd: 8b 03     mov    eax,DWORD PTR [ebx]
80484cf: 83 f8 ff   cmp    eax,0xffffffff
80484d2: 75 f4     jne   80484c8 <__do_global_ctors_aux+0x18>
80484d4: 83 c4 04   add    esp,0x4
80484d7: 5b        pop    ebx
80484d8: 5d        pop    ebp
80484d9: c3        ret
80484da: 90          nop
80484db: 90          nop

```

Disassembliamo la sezione .init

**\$ objdump -d -j .init prova**

prova: file format elf32-i386

Disassembly of section .init:

```

08048290 <_init>:
8048290: 55          push   %ebp
8048291: 89 e5      mov    %esp,%ebp
8048293: 53          push   %ebx
8048294: 83 ec 04   sub    $0x4,%esp
8048297: e8 00 00 00 00 call  804829c <_init+0xc>
804829c: 5b        pop    %ebx
804829d: 81 c3 fc 13 00 00 add   $0x13fc,%ebx
80482a3: 8b 93 fc ff ff ff mov   -0x4(%ebx),%edx
80482a9: 85 d2     test  %edx,%edx
80482ab: 74 05     je    80482b2 <_init+0x22>
80482ad: e8 1e 00 00 00 call  80482d0 <__gmon_start__@plt>
80482b2: e8 e9 00 00 00 call  80483a0 <frame_dummy>
80482b7: e8 f4 01 00 00 call  80484b0 <__do_global_ctors_aux>
80482bc: 58        pop    %eax
80482bd: 5b        pop    %ebx
80482be: c9        leave
80482bf: c3        ret

```

**\$ objdump -d -j .fini prova**

prova: file format elf32-i386

Disassembly of section .fini:

```

080484dc <_fini>:
80484dc:    55                push   %ebp
80484dd:    89 e5            mov    %esp,%ebp
80484df:    53                push   %ebx
80484e0:    83 ec 04         sub   $0x4,%esp
80484e3:    e8 00 00 00 00   call  80484e8 <_fini+0xc>
80484e8:    5b                pop    %ebx
80484e9:    81 c3 b0 11 00 00 add   $0x11b0,%ebx
80484ef:    e8 4c fe ff ff   call  8048340 <__do_global_dtors_aux>
80484f4:    59                pop    %ecx
80484f5:    5b                pop    %ebx
80484f6:    c9                leave
80484f7:    c3                ret

```

L'entry point nell'header ELF punta all'indirizzo 0x08048310, che corrisponde alla funzione `_start`, la prima quindi ad essere eseguita. Si tratta in realtà di un wrapper per la funzione della libc `<__libc_start_main>`.

```

extern int BP_SYM (<__libc_start_main>) (int (*main) (int, char**, char**),
    int argc,
    char *__unbounded *__unbounded ubp_av,
    void (*init) (void),
    void (*fini) (void),
    void (*rtld_fini) (void),
    void *__unbounded stack_end)
__attribute__((noreturn));

```

Gli argomenti passati a questa funzione sono inseriti nello stack e sono evidenziati in grassetto nel listato:

0x80483dd	main
<b>%esi</b>	argc
<b>%ecx</b>	argv
0x8048450	<__libc_csu_init> ( <code>_init</code> )
0x8048440	<__libc_csu_fini> ( <code>_fini</code> )
<b>%edx</b>	rtld_fini (inizializzato con <code>atexit()</code> )
<b>%esp</b>	stack_end allineato a 4k
<b>%eax</b>	0

I valori dei registri che `start` inserisce nello stack, sono stati predisposti dal kernel tramite la funzione `“execve”` e la system call `“sys_execve”`; a questo punto i parametri passati dalla applicazione al kernel sono memorizzati in alcuni registri:

- `%ebx` contiene il puntatore alla stringa del nome del programma
- `%ecx` contiene il puntatore all'array `argv`
- `%edx` contiene il puntatore all'array delle variabili d'ambiente.

La system call handler chiama poi `search_binary_handler()`, la routine che sceglie il loader adatto al formato di file; usa allo scopo la struttura `“struct linux_binfmt”` che contiene un puntatore a funzione per ogni formato di file binario. Nel caso di file ELF, usa `load_elf_binary()`.

Questa funzione inizializza le strutture dati del kernel necessarie per leggere il file ELF e la struttura dati che contiene: dimensione del codice, inizio del segmento dati, inizio del segmento stack, ecc. Alloca le pagine in user mode per il processo e vi copia `argv` e variabili di ambiente. Infine `argc`, il puntatore ad `argv` e quello alle variabili di ambiente sono inseriti nello user-mode stack da `create_elf_tables()`, e

start\_thread() avvia l'esecuzione del programma.

Quando \_start assume il controllo dell'esecuzione, lo stack frame appare così:

argc
argv pointer
envp pointer

Le prime istruzioni di \_start ricavano dallo stack le seguenti informazioni:

```
08048310 <_start>:
8048310: 31 ed          xor    ebp,ebp    ← azzera ebp
8048312: 5e            pop    esi        ← ottiene argc
8048313: 89 e1        mov    ecx,esp    ← ottiene *argv
```

La <\_\_libc\_start\_main> compie alcune operazioni fondamentali: inizializza la libreria C e l'ambiente thread, Infine chiama main.

Funzione di inizializzazione \_init.

La parte importante del codice è questa:

```
80482ad: e8 1e 00 00 00 call 80482d0 <__gmon_start__@plt>
80482b2: e8 e9 00 00 00 call 80483a0 <frame_dummy>
80482b7: e8 f4 01 00 00 call 80484b0 <__do_global_ctors_aux>
```

La funzione call\_gmon\_start inizializza il sistema di profiling gmon, abilitato quando i binari sono compilati con l'opzione -pg; esso crea un output che viene usato da gprof. Il sistema di profilazione rileva il tempo speso dal programma in ogni routine. call\_gmon\_start trova l'ultima voce della GOT, nota come \_\_gmon\_start\_\_, e, se il valore non è NULL, passa il controllo all'indirizzo in essa contenuto. L'elemento \_\_gmon\_start\_\_ punta alla funzione di inizializzazione gmon, che inizia a immagazzinare informazioni di profilazione e registra una funzione di pulizia con atexit().

La <\_\_do\_global\_ctors\_aux> ha il compito di processare tutte le voci di costruttori della sezione .ctors

Nella sezione .fini, la funzione \_fini ha lo scopo di chiamare la <\_\_do\_global\_dtors\_aux> che processa i distruttori presenti nella sezione .dtors. Nel nostro esempio le sezioni .ctors e .dtors sono vuote.

La breve funzione <\_\_i686.get\_pc\_thunk.bx> viene usata nell'architettura x86 per il codice indipendente dalla posizione (PIC code), cioè il codice che viene caricato ed eseguito senza modifiche in diversi spazi di indirizzi, presente nelle librerie condivise. Altre architetture di CPU hanno un registro dedicato al PIC code. In x86 la funzione <\_\_i686.get\_pc\_thunk.bx> carica la posizione del codice nel registro %ebx; ciò permette di accedere agli oggetti condivisi tramite un offset dal registro stesso.

## SEZIONE .DATA e .BSS

La stessa cosa è possibile per la sezione .data

```
$ objdump -s prova | grep -w .data -A 1
```

```
Contents of section .data:
80496b4 00000000 00000000 03000000 .....
```

```
$ objdump -M intel -d -j .data prova
```

```
prova: file format elf32-i386
```

```
Disassembly of section .data:
```

```
080496b4 <__data_start>:
80496b4: 00 00          add    BYTE PTR [eax],al
      ...

080496b8 <__dso_handle>:
80496b8: 00 00 00 00          ....

080496bc <var_globale_1>:
80496bc: 03 00 00 00          ....
      ....
```

Si vede che la sezione contiene solo la variabile inizializzata “var\_globale\_1” con valore 3. La variabile globale non inizializzata “var\_globale\_2” si trova nella sezione .bss (Block Started by Symbol) e viene inizializzata dal compilatore a 0 che, per variabili di tipo char, significa carattere NULL.

```
$ objdump -M intel -d -j .bss prova
```

```
prova: file format elf32-i386
```

```
Disassembly of section .bss:
```

```
080496c0 <completed.6971>:
80496c0: 00 00 00 00          ....

080496c4 <dtor_idx.6973>:
80496c4: 00 00 00 00          ....

080496c8 <var_globale_2>:
80496c8: 00 00 00 00          ....
```

## TABELLA DEI SIMBOLI

La tabella dei simboli mette in correlazione il nome di un simbolo (funzione non esterna, variabile) con un indirizzo (campo value nella tabella). Per visualizzarla:

```
$ objdump -t prova
```

```
prova: file format elf32-i386
```

```
SYMBOL TABLE:
08048134 l d .interp 00000000 .interp
```

08048148	l	d	.note.ABI-tag	00000000	.note.ABI-tag
08048168	l	d	.gnu.hash	00000000	.gnu.hash
08048188	l	d	.dynsym	00000000	.dynsym
080481d8	l	d	.dynstr	00000000	.dynstr
08048224	l	d	.gnu.version	00000000	.gnu.version
08048230	l	d	.gnu.version_r	00000000	.gnu.version_r
08048250	l	d	.rel.dyn	00000000	.rel.dyn
08048258	l	d	.rel.plt	00000000	.rel.plt
08048270	l	d	.init	00000000	.init
080482a0	l	d	.plt	00000000	.plt
080482e0	l	d	.text	00000000	.text
0804845c	l	d	.fini	00000000	.fini
08048478	l	d	.rodata	00000000	.rodata
0804848c	l	d	.eh_frame_hdr	00000000	.eh_frame_hdr
080484b0	l	d	.eh_frame	00000000	.eh_frame
08049524	l	d	.ctors	00000000	.ctors
0804952c	l	d	.dtors	00000000	.dtors
08049534	l	d	.jcr	00000000	.jcr
08049538	l	d	.dynamic	00000000	.dynamic
08049600	l	d	.got	00000000	.got
08049604	l	d	.got.plt	00000000	.got.plt
0804961c	l	d	.data	00000000	.data
08049624	l	d	.bss	00000000	.bss
00000000	l	d	.comment	00000000	.comment
00000000	l	d	.debug_aranges	00000000	.debug_aranges
00000000	l	d	.debug_pubnames	00000000	.debug_pubnames
00000000	l	d	.debug_info	00000000	.debug_info
00000000	l	d	.debug_abbrev	00000000	.debug_abbrev
00000000	l	d	.debug_line	00000000	.debug_line
00000000	l	d	.debug_str	00000000	.debug_str
00000000	l	d	.debug_loc	00000000	.debug_loc
00000000	l	df	*ABS*	00000000	init.c
00000000	l	df	*ABS*	00000000	initfini.c
00000000	l	df	*ABS*	00000000	crtstuff.c
08049524	l	0	.ctors	00000000	__CTOR_LIST__
0804952c	l	0	.dtors	00000000	__DTOR_LIST__
08049534	l	0	.jcr	00000000	__JCR_LIST__
08048310	l	F	.text	00000000	__do_global_dtors_aux
08049624	l	0	.bss	00000001	completed.6971
08049628	l	0	.bss	00000004	dtor_idx.6973
08048370	l	F	.text	00000000	frame_dummy
00000000	l	df	*ABS*	00000000	crtstuff.c
08049528	l	0	.ctors	00000000	__CTOR_END__
08048520	l	0	.eh_frame	00000000	__FRAME_END__
08049534	l	0	.jcr	00000000	__JCR_END__
08048430	l	F	.text	00000000	__do_global_ctors_aux
00000000	l	df	*ABS*	00000000	initfini.c
00000000	l	df	*ABS*	00000000	prova.c
08049604	l	0	.got.plt	00000000	.hidden __GLOBAL_OFFSET_TABLE__
08049524	l		.ctors	00000000	.hidden __init_array_end
08049524	l		.ctors	00000000	.hidden __init_array_start
08049538	l	0	.dynamic	00000000	.hidden __DYNAMIC
0804961c	w		.data	00000000	data_start
080483c0	g	F	.text	00000005	__libc_csu_fini
080482e0	g	F	.text	00000000	__start
00000000	w		*UND*	00000000	__gmon_start__
00000000	w		*UND*	00000000	__Jv_RegisterClasses
08048478	g	0	.rodata	00000004	__fp_hw
0804845c	g	F	.fini	00000000	__fini
00000000		F	*UND*	00000000	__libc_start_main@@GLIBC_2.0
0804847c	g	0	.rodata	00000004	__IO_stdin_used
0804961c	g		.data	00000000	__data_start
08049620	g	0	.data	00000000	.hidden __dso_handle
08049530	g	0	.dtors	00000000	.hidden __DTOR_END__

```

080483d0 g    F .text      0000005a      __libc_csu_init
00000000    F *UND*      00000000      printf@@GLIBC_2.0
08049624 g    *ABS*      00000000      __bss_start
0804962c g    *ABS*      00000000      _end
08049624 g    *ABS*      00000000      _edata
0804842a g    F .text      00000000      .hidden __i686.get_pc_thunk.bx
08048394 g    F .text      00000028      main
08048270 g    F .init      00000000      _init

```

oppure

## \$ readelf -s prova

Symbol table '.dynsym' contains 6 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	00000000	0	NOTYPE	WEAK	DEFAULT	UND	__gmon_start__
2:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	__libc_start_main@GLIBC_2.0 (2)
3:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	printf@GLIBC_2.0 (2)
4:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	puts@GLIBC_2.0 (2)
5:	080484fc	4	OBJECT	GLOBAL	DEFAULT	14	_IO_stdin_used

Symbol table '.symtab' contains 77 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	08048134	0	SECTION	LOCAL	DEFAULT	1	
2:	08048148	0	SECTION	LOCAL	DEFAULT	2	
3:	08048168	0	SECTION	LOCAL	DEFAULT	3	
4:	08048188	0	SECTION	LOCAL	DEFAULT	4	
5:	080481e8	0	SECTION	LOCAL	DEFAULT	5	
6:	0804823a	0	SECTION	LOCAL	DEFAULT	6	
7:	08048248	0	SECTION	LOCAL	DEFAULT	7	
8:	08048268	0	SECTION	LOCAL	DEFAULT	8	
9:	08048270	0	SECTION	LOCAL	DEFAULT	9	
10:	08048290	0	SECTION	LOCAL	DEFAULT	10	
11:	080482c0	0	SECTION	LOCAL	DEFAULT	11	
12:	08048310	0	SECTION	LOCAL	DEFAULT	12	
13:	080484dc	0	SECTION	LOCAL	DEFAULT	13	
14:	080484f8	0	SECTION	LOCAL	DEFAULT	14	
15:	08048544	0	SECTION	LOCAL	DEFAULT	15	
16:	08048560	0	SECTION	LOCAL	DEFAULT	16	
17:	080495b8	0	SECTION	LOCAL	DEFAULT	17	
18:	080495c0	0	SECTION	LOCAL	DEFAULT	18	
19:	080495c8	0	SECTION	LOCAL	DEFAULT	19	
20:	080495cc	0	SECTION	LOCAL	DEFAULT	20	
21:	08049694	0	SECTION	LOCAL	DEFAULT	21	
22:	08049698	0	SECTION	LOCAL	DEFAULT	22	
23:	080496b4	0	SECTION	LOCAL	DEFAULT	23	
24:	080496c0	0	SECTION	LOCAL	DEFAULT	24	
25:	00000000	0	SECTION	LOCAL	DEFAULT	25	
26:	00000000	0	SECTION	LOCAL	DEFAULT	26	
27:	00000000	0	SECTION	LOCAL	DEFAULT	27	
28:	00000000	0	SECTION	LOCAL	DEFAULT	28	
29:	00000000	0	SECTION	LOCAL	DEFAULT	29	
30:	00000000	0	SECTION	LOCAL	DEFAULT	30	
31:	00000000	0	SECTION	LOCAL	DEFAULT	31	
32:	00000000	0	FILE	LOCAL	DEFAULT	ABS	init.c
33:	00000000	0	FILE	LOCAL	DEFAULT	ABS	initfini.c
34:	00000000	0	FILE	LOCAL	DEFAULT	ABS	crtstuff.c
35:	080495b8	0	OBJECT	LOCAL	DEFAULT	17	__CTOR_LIST__

```

36: 080495c0    0 OBJECT LOCAL DEFAULT 18 __DTOR_LIST__
37: 080495c8    0 OBJECT LOCAL DEFAULT 19 __JCR_LIST__
38: 08048340    0 FUNC LOCAL DEFAULT 12 __do_global_dtors_aux
39: 080496c0    1 OBJECT LOCAL DEFAULT 24 completed.6971
40: 080496c4    4 OBJECT LOCAL DEFAULT 24 dtor_idx.6973
41: 080483a0    0 FUNC LOCAL DEFAULT 12 frame_dummy
42: 00000000    0 FILE LOCAL DEFAULT ABS crtstuff.c
43: 080495bc    0 OBJECT LOCAL DEFAULT 17 __CTOR_END__
44: 080485b4    0 OBJECT LOCAL DEFAULT 16 __FRAME_END__
45: 080495c8    0 OBJECT LOCAL DEFAULT 19 __JCR_END__
46: 080484b0    0 FUNC LOCAL DEFAULT 12 __do_global_ctors_aux
47: 00000000    0 FILE LOCAL DEFAULT ABS initfini.c
48: 00000000    0 FILE LOCAL DEFAULT ABS prova.c
49: 08049698    0 OBJECT LOCAL HIDDEN 22 _GLOBAL_OFFSET_TABLE_
50: 080495b8    0 NOTYPE LOCAL HIDDEN 17 __init_array_end
51: 080495b8    0 NOTYPE LOCAL HIDDEN 17 __init_array_start
52: 080495cc    0 OBJECT LOCAL HIDDEN 20 _DYNAMIC
53: 080496b4    0 NOTYPE WEAK DEFAULT 23 data_start
54: 08048440    5 FUNC GLOBAL DEFAULT 12 __libc_csu_fini
55: 08048310    0 FUNC GLOBAL DEFAULT 12 _start
56: 00000000    0 NOTYPE WEAK DEFAULT UND __gmon_start__
57: 00000000    0 NOTYPE WEAK DEFAULT UND _Jv_RegisterClasses
58: 080484f8    4 OBJECT GLOBAL DEFAULT 14 _fp_hw
59: 080484dc    0 FUNC GLOBAL DEFAULT 13 _fini
60: 00000000    0 FUNC GLOBAL DEFAULT UND __libc_start_main@@GLIBC_
61: 080483c4    25 FUNC GLOBAL DEFAULT 12 funzione_vuota
62: 080484fc    4 OBJECT GLOBAL DEFAULT 14 _IO_stdin_used
63: 080496b4    0 NOTYPE GLOBAL DEFAULT 23 __data_start
64: 080496bc    4 OBJECT GLOBAL DEFAULT 23 var_globale_1
65: 080496b8    0 OBJECT GLOBAL HIDDEN 23 __dso_handle
66: 080495c4    0 OBJECT GLOBAL HIDDEN 18 __DTOR_END__
67: 08048450    90 FUNC GLOBAL DEFAULT 12 __libc_csu_init
68: 00000000    0 FUNC GLOBAL DEFAULT UND printf@@GLIBC_2.0
69: 080496c0    0 NOTYPE GLOBAL DEFAULT ABS __bss_start
70: 080496c8    4 OBJECT GLOBAL DEFAULT 24 var_globale_2
71: 080496cc    0 NOTYPE GLOBAL DEFAULT ABS _end
72: 00000000    0 FUNC GLOBAL DEFAULT UND puts@@GLIBC_2.0
73: 080496c0    0 NOTYPE GLOBAL DEFAULT ABS _edata
74: 080484aa    0 FUNC GLOBAL HIDDEN 12 __i686.get_pc_thunk.bx
75: 080483dd    96 FUNC GLOBAL DEFAULT 12 main
76: 08048290    0 FUNC GLOBAL DEFAULT 10 _init

```

Ricordando dalla tabella degli header di sezione che la tabella dei simboli inizia all'offset 0x10a0 e quella delle stringhe inizia all'offset 0x1570,

```

[33] .symtab          SYMTAB          00000000 0010a0 0004d0 10      34 53 4
[34] .strtab           STRTAB          00000000 001570 00024a 00       0  0 1

```

possiamo visualizzarle entrambe con

**\$ hexdump -C prova | tail -n 115**

```

000010a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000010b0 00 00 00 00 34 81 04 08 00 00 00 00 03 00 01 00 |....4.....|
000010c0 00 00 00 00 48 81 04 08 00 00 00 00 03 00 02 00 |....H.....|
000010d0 00 00 00 00 68 81 04 08 00 00 00 00 03 00 03 00 |....h.....|
000010e0 00 00 00 00 88 81 04 08 00 00 00 00 03 00 04 00 |.....|
000010f0 00 00 00 00 e8 81 04 08 00 00 00 00 03 00 05 00 |.....|
00001100 00 00 00 00 3a 82 04 08 00 00 00 00 03 00 06 00 |....:.....|
00001110 00 00 00 00 48 82 04 08 00 00 00 00 03 00 07 00 |....H.....|

```

00001120	00 00 00 00 68 82 04 08	00 00 00 00 03 00 08 00	....h.....
00001130	00 00 00 00 70 82 04 08	00 00 00 00 03 00 09 00	....p.....
00001140	00 00 00 00 90 82 04 08	00 00 00 00 03 00 0a 00	.....
00001150	00 00 00 00 c0 82 04 08	00 00 00 00 03 00 0b 00	.....
00001160	00 00 00 00 10 83 04 08	00 00 00 00 03 00 0c 00	.....
00001170	00 00 00 00 dc 84 04 08	00 00 00 00 03 00 0d 00	.....
00001180	00 00 00 00 f8 84 04 08	00 00 00 00 03 00 0e 00	.....
00001190	00 00 00 00 44 85 04 08	00 00 00 00 03 00 0f 00	....D.....
000011a0	00 00 00 00 60 85 04 08	00 00 00 00 03 00 10 00	....`.....
000011b0	00 00 00 00 b8 95 04 08	00 00 00 00 03 00 11 00	.....
000011c0	00 00 00 00 c0 95 04 08	00 00 00 00 03 00 12 00	.....
000011d0	00 00 00 00 c8 95 04 08	00 00 00 00 03 00 13 00	.....
000011e0	00 00 00 00 cc 95 04 08	00 00 00 00 03 00 14 00	.....
000011f0	00 00 00 00 94 96 04 08	00 00 00 00 03 00 15 00	.....
00001200	00 00 00 00 98 96 04 08	00 00 00 00 03 00 16 00	.....
00001210	00 00 00 00 b4 96 04 08	00 00 00 00 03 00 17 00	.....
00001220	00 00 00 00 c0 96 04 08	00 00 00 00 03 00 18 00	.....
00001230	00 00 00 00 00 00 00 00	00 00 00 00 03 00 19 00	.....
00001240	00 00 00 00 00 00 00 00	00 00 00 00 03 00 1a 00	.....
00001250	00 00 00 00 00 00 00 00	00 00 00 00 03 00 1b 00	.....
00001260	00 00 00 00 00 00 00 00	00 00 00 00 03 00 1c 00	.....
00001270	00 00 00 00 00 00 00 00	00 00 00 00 03 00 1d 00	.....
00001280	00 00 00 00 00 00 00 00	00 00 00 00 03 00 1e 00	.....
00001290	00 00 00 00 00 00 00 00	00 00 00 00 03 00 1f 00	.....
000012a0	01 00 00 00 00 00 00 00	00 00 00 00 04 00 f1 ff	.....
000012b0	08 00 00 00 00 00 00 00	00 00 00 00 04 00 f1 ff	.....
000012c0	13 00 00 00 00 00 00 00	00 00 00 00 04 00 f1 ff	.....
000012d0	1e 00 00 00 b8 95 04 08	00 00 00 00 01 00 11 00	.....
000012e0	2c 00 00 00 c0 95 04 08	00 00 00 00 01 00 12 00	,.....
000012f0	3a 00 00 00 c8 95 04 08	00 00 00 00 01 00 13 00	:.....
00001300	47 00 00 00 40 83 04 08	00 00 00 00 02 00 0c 00	G...@.....
00001310	5d 00 00 00 c0 96 04 08	01 00 00 00 01 00 18 00	].....
00001320	6c 00 00 00 c4 96 04 08	04 00 00 00 01 00 18 00	l.....
00001330	7a 00 00 00 a0 83 04 08	00 00 00 00 02 00 0c 00	z.....
00001340	13 00 00 00 00 00 00 00	00 00 00 00 04 00 f1 ff	.....
00001350	86 00 00 00 bc 95 04 08	00 00 00 00 01 00 11 00	.....
00001360	93 00 00 00 b4 85 04 08	00 00 00 00 01 00 10 00	.....
00001370	a1 00 00 00 c8 95 04 08	00 00 00 00 01 00 13 00	.....
00001380	ad 00 00 00 b0 84 04 08	00 00 00 00 02 00 0c 00	.....
00001390	08 00 00 00 00 00 00 00	00 00 00 00 04 00 f1 ff	.....
000013a0	c3 00 00 00 00 00 00 00	00 00 00 00 04 00 f1 ff	.....
000013b0	cb 00 00 00 98 96 04 08	00 00 00 00 01 02 16 00	.....
000013c0	e1 00 00 00 b8 95 04 08	00 00 00 00 00 02 11 00	.....
000013d0	f2 00 00 00 b8 95 04 08	00 00 00 00 00 02 11 00	.....
000013e0	05 01 00 00 cc 95 04 08	00 00 00 00 01 02 14 00	.....
000013f0	0e 01 00 00 b4 96 04 08	00 00 00 00 20 00 17 00	.....
00001400	19 01 00 00 40 84 04 08	05 00 00 00 12 00 0c 00	....@.....
00001410	29 01 00 00 10 83 04 08	00 00 00 00 12 00 0c 00	).....
00001420	30 01 00 00 00 00 00 00	00 00 00 00 20 00 00 00	0.....
00001430	3f 01 00 00 00 00 00 00	00 00 00 00 20 00 00 00	?.....
00001440	53 01 00 00 f8 84 04 08	04 00 00 00 11 00 0e 00	S.....
00001450	5a 01 00 00 dc 84 04 08	00 00 00 00 12 00 0d 00	Z.....
00001460	60 01 00 00 00 00 00 00	00 00 00 00 12 00 00 00	`.....
00001470	7d 01 00 00 c4 83 04 08	19 00 00 00 12 00 0c 00	}.....
00001480	8c 01 00 00 fc 84 04 08	04 00 00 00 11 00 0e 00	.....
00001490	9b 01 00 00 b4 96 04 08	00 00 00 00 10 00 17 00	.....
000014a0	a8 01 00 00 bc 96 04 08	04 00 00 00 11 00 17 00	.....
000014b0	b6 01 00 00 b8 96 04 08	00 00 00 00 11 02 17 00	.....
000014c0	c3 01 00 00 c4 95 04 08	00 00 00 00 11 02 12 00	.....
000014d0	d0 01 00 00 50 84 04 08	5a 00 00 00 12 00 0c 00	....P...Z.....
000014e0	e0 01 00 00 00 00 00 00	00 00 00 00 12 00 00 00	.....
000014f0	f2 01 00 00 c0 96 04 08	00 00 00 00 10 00 f1 ff	.....
00001500	fe 01 00 00 c8 96 04 08	04 00 00 00 11 00 18 00	.....
00001510	0c 02 00 00 cc 96 04 08	00 00 00 00 10 00 f1 ff	.....

```

00001520 11 02 00 00 00 00 00 00 00 00 00 00 12 00 00 00 |.....|
00001530 21 02 00 00 c0 96 04 08 00 00 00 00 10 00 f1 ff |!.....|
00001540 28 02 00 00 aa 84 04 08 00 00 00 00 12 02 0c 00 |(.....|
00001550 3f 02 00 00 dd 83 04 08 60 00 00 00 12 00 0c 00 |?.....`.....|
00001560 44 02 00 00 90 82 04 08 00 00 00 00 12 00 0a 00 |D.....|

```

```

00001570 00 69 6e 69 74 2e 63 00 69 6e 69 74 66 69 6e 69 |.init.c.initfini|
00001580 2e 63 00 63 72 74 73 74 75 66 66 2e 63 00 5f 5f |.c.crtstuff.c.__|
00001590 43 54 4f 52 5f 4c 49 53 54 5f 5f 00 5f 5f 44 54 |CTOR_LIST__.__DT|
000015a0 4f 52 5f 4c 49 53 54 5f 5f 00 5f 5f 4a 43 52 5f |OR_LIST__.__JCR_|
000015b0 4c 49 53 54 5f 5f 00 5f 5f 64 6f 5f 67 6c 6f 62 |LIST__.__do_glob|
000015c0 61 6c 5f 64 74 6f 72 73 5f 61 75 78 00 63 6f 6d |al_dtors_aux.com|
000015d0 70 6c 65 74 65 64 2e 36 39 37 31 00 64 74 6f 72 |pleted.6971.dtor|
000015e0 5f 69 64 78 2e 36 39 37 33 00 66 72 61 6d 65 5f |__idx.6973.frame_|
000015f0 64 75 6d 6d 79 00 5f 5f 43 54 4f 52 5f 45 4e 44 |dummy.__CTOR_END|
00001600 5f 5f 00 5f 5f 46 52 41 4d 45 5f 45 4e 44 5f 5f |__.__FRAME_END__|
00001610 00 5f 5f 4a 43 52 5f 45 4e 44 5f 5f 00 5f 5f 64 |__.__JCR_END__.__d|
00001620 6f 5f 67 6c 6f 62 61 6c 5f 63 74 6f 72 73 5f 61 |o_global_ctors_a|
00001630 75 78 00 70 72 6f 76 61 2e 63 00 5f 47 4c 4f 42 |ux.prova.c._GLOB|
00001640 41 4c 5f 4f 46 46 53 45 54 5f 54 41 42 4c 45 5f |AL_OFFSET_TABLE_|
00001650 00 5f 5f 69 6e 69 74 5f 61 72 72 61 79 5f 65 6e |.__init_array_en|
00001660 64 00 5f 5f 69 6e 69 74 5f 61 72 72 61 79 5f 73 |d.__init_array_s|
00001670 74 61 72 74 00 5f 44 59 4e 41 4d 49 43 00 64 61 |tart._DYNAMIC.da|
00001680 74 61 5f 73 74 61 72 74 00 5f 5f 6c 69 62 63 5f |ta_start.__libc_|
00001690 63 73 75 5f 66 69 6e 69 00 5f 73 74 61 72 74 00 |csu_fini.__start_|
000016a0 5f 5f 67 6d 6f 6e 5f 73 74 61 72 74 5f 5f 00 5f |__gmon_start__._|
000016b0 4a 76 5f 52 65 67 69 73 74 65 72 43 6c 61 73 73 |Jv_RegisterClass|
000016c0 65 73 00 5f 66 70 5f 68 77 00 5f 66 69 6e 69 00 |es._fp_hw._fini_|
000016d0 5f 5f 6c 69 62 63 5f 73 74 61 72 74 5f 6d 61 69 |__libc_start_mai|
000016e0 6e 40 40 47 4c 49 42 43 5f 32 2e 30 00 66 75 6e |n@@GLIBC_2.0.fun|
000016f0 7a 69 6f 6e 65 5f 76 75 6f 74 61 00 5f 49 4f 5f |zione_vuota._IO_|
00001700 73 74 64 69 6e 5f 75 73 65 64 00 5f 5f 64 61 74 |stdin_used.__dat|
00001710 61 5f 73 74 61 72 74 00 76 61 72 5f 67 6c 6f 62 |a_start.var_glob|
00001720 61 6c 65 5f 31 00 5f 5f 64 73 6f 5f 68 61 6e 64 |ale_1.__dso_hand|
00001730 6c 65 00 5f 5f 44 54 4f 52 5f 45 4e 44 5f 5f 00 |le.__DTOR_END__._|
00001740 5f 5f 6c 69 62 63 5f 63 73 75 5f 69 6e 69 74 00 |__libc_csu_init_|
00001750 70 72 69 6e 74 66 40 40 47 4c 49 42 43 5f 32 2e |printf@@GLIBC_2_|
00001760 30 00 5f 5f 62 73 73 5f 73 74 61 72 74 00 76 61 |0.__bss_start.va|
00001770 72 5f 67 6c 6f 62 61 6c 65 5f 32 00 5f 65 6e 64 |r_globale_2._end|
00001780 00 70 75 74 73 40 40 47 4c 49 42 43 5f 32 2e 30 |.puts@@GLIBC_2.0|
00001790 00 5f 65 64 61 74 61 00 5f 5f 69 36 38 36 2e 67 |._edata._i686.g|
000017a0 65 74 5f 70 63 5f 74 68 75 6e 6b 2e 62 78 00 6d |et_pc_thunk.bx.m|
000017b0 61 69 6e 00 5f 69 6e 69 74 00 |ain._init.|
000017ba

```

Le funzioni definite nel file, come **funzione\_vuota** evidenziata in grassetto, hanno un indirizzo nella tabella dei simboli.

Le funzioni di libreria, come printf, non hanno un indirizzo

```
67: 00000000      0 FUNC      GLOBAL DEFAULT UND printf@@GLIBC_2.0
```

in quanto non sono definite entro il programma. Per queste ultime si usa un'altra tabella, la PLT.

### Analisi di una voce della tabella dei simboli

Analizziamo una voce della tabella dei simboli come si presenta in esadecimale. I campi della tabella sono

```
typedef struct {
    Elf32_Word      st_name;      /* Symbol name (string tbl index) */
    Elf32_Addr     st_value;     /* Symbol value */
}
```

```

Elf32_Word      st_size;      /* Symbol size */
unsigned char   st_info;      /* Symbol type and binding */
unsigned char   st_other;     /* Symbol visibility */
Elf32_Section   st_shndx;     /* Section index */
} Elf32_Sym;

```

Questa è una voce da analizzare

```
000012d0  1e 00 00 00 b8 95 04 08 00 00 00 00 01 00 11 00 |.....|
```

con la visualizzazione ottenuta da readelf:

```

Num:      Value  Size Type   Bind  Vis      Ndx Name
35: 080495b8    0 OBJECT LOCAL  DEFAULT 17 __CTOR_LIST__

```

Vediamo il significato di ogni elemento della struttura

```
000012d0  1e 00 00 00 b8 95 04 08 00 00 00 00 01 00 11 00 |.....|
```

- **1e 00 00 00** (ricordiamo che i numeri sono little endian; quindi vale 0x001e cioè 30 in decimale) è il campo `st_name`, che rappresenta una entry nella tabella dei simboli qui evidenziata nell'estratto della tabella stessa

```

00001570  00 69 6e 69 74 2e 63 00 69 6e 69 74 66 69 6e 69 |.init.c.initfini|
00001580  2e 63 00 63 72 74 73 74 75 66 66 2e 63 00 5f 5f |.c.crtstuff.c.__|
00001590  43 54 4f 52 5f 4c 49 53 54 5f 5f 00 5f 5f 44 54 |CTOR_LIST__.__DT|

```

- La stringa che rappresenta il nome del simbolo inizia alla posizione 30 e termina al simbolo '\0' (terminatore di stringa): si può perciò leggere il nome `__CTOR_LIST__`.

```
000012d0  1e 00 00 00 b8 95 04 08 00 00 00 00 01 00 11 00 |.....|
```

- **b8 95 04 08** è il campo `st_value`, che in questo caso rappresenta l'indirizzo (sempre in formato little endian) 0x080495b8

```
000012d0  1e 00 00 00 b8 95 04 08 00 00 00 00 01 00 11 00 |.....|
```

- **00 00 00 00** il campo `st_size`, che rappresenta la dimensione dell'oggetto

```
000012d0  1e 00 00 00 b8 95 04 08 00 00 00 00 01 00 11 00 |.....|
```

- **01** è il campo `st_info`, che contiene tipo e binding del simbolo; in questo caso i valori si ricavano tramite le macro

```

#define ELF32_ST_BIND(val) (((unsigned char) (val)) >> 4)
#define ELF32_ST_TYPE(val) ((val) & 0xf)
#define ELF32_ST_INFO(bind, type) (((bind) << 4) + ((type) & 0xf))

```

e, nel caso specifico, forniscono i valori di tipo = 1 (oggetto) e binding = 0 (local).

```
000012d0  1e 00 00 00 b8 95 04 08 00 00 00 00 01 00 11 00 |.....|
```

- **00** è il campo `st_other`, che rappresenta la visibilità, in questo caso 0 (default)

```
000012d0 1e 00 00 00 b8 95 04 08 00 00 00 00 01 00 11 00 |.....|
```

- **11 00** è il campo `st_shndx` che rappresenta l'indice della tabella delle sezioni; in questo caso il valore si riferisce alla sezione n° 17 (0x11) e cioè `.ctors`

```
[17] .ctors          PROGBITS          080495b8 0005b8 000008 00 WA 0 0 4
```

## SEGMENTI

Per visualizzarli:

### \$ readelf -l prova

```
Elf file type is EXEC (Executable file)
Entry point 0x8048310
There are 8 program headers, starting at offset 52
```

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
PHDR	0x000034	0x08048034	0x08048034	0x00100	0x00100	R E	0x4
INTERP	0x000134	0x08048134	0x08048134	0x00013	0x00013	R	0x1
[Requesting program interpreter: /lib/ld-linux.so.2]							
LOAD	0x000000	0x08048000	0x08048000	0x005b8	0x005b8	R E	0x1000
LOAD	0x0005b8	0x080495b8	0x080495b8	0x00108	0x00114	RW	0x1000
DYNAMIC	0x0005cc	0x080495cc	0x080495cc	0x000c8	0x000c8	RW	0x4
NOTE	0x000148	0x08048148	0x08048148	0x00020	0x00020	R	0x4
GNU_EH_FRAME	0x000544	0x08048544	0x08048544	0x0001c	0x0001c	R	0x4
GNU_STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RW	0x4

Section to Segment mapping:

```
Segment Sections...
00
01      .interp
02      .interp .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version
.gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame_hdr
.eh_frame
03      .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
04      .dynamic
05      .note.ABI-tag
06      .eh_frame_hdr
07
```

Si può vedere in quale segmento sono mappate le sezioni e qual è l'indirizzo iniziale di ogni segmento (VirtAddr).

Lo stack non ha valori impostati perché è il kernel a decidere il suo indirizzo iniziale e la dimensione.

Per vedere il layout dei segmenti del processo si può cercare nella directory `/proc/<pid del processo>/maps`. Il problema è che questo file vive finché vive il processo, ed il nostro ha una vita molto breve; bisogna congelarlo con il debugger

### \$ gdb -q prova

```
(gdb) break main
```

## Breakpoint 1 at 0x80483e0

(gdb) run

Starting program: /home/andrea/Scrivania/prova

Breakpoint 1, 0x080483e0 in main ()

(gdb)

ora che il programma è bloccato su main, da un altro terminale

\$ cat /proc/\$(pgrep prova)/maps

```
08048000-08049000 r-xp 00000000 08:06 132605 /home/prova
08049000-0804a000 rw-p 00000000 08:06 132605 /home/prova
b75a2000-b75a3000 rw-p 00000000 00:00 0
b75a3000-b76fd000 r-xp 00000000 08:05 246013 /lib/i686/libc-2.10.1.so
b76fd000-b76fe000 ---p 0015a000 08:05 246013 /lib/i686/libc-2.10.1.so
b76fe000-b7700000 r--p 0015a000 08:05 246013 /lib/i686/libc-2.10.1.so
b7700000-b7701000 rw-p 0015c000 08:05 246013 /lib/i686/libc-2.10.1.so
b7701000-b7704000 rw-p 00000000 00:00 0
b771e000-b771f000 rw-p 00000000 00:00 0
b771f000-b773b000 r-xp 00000000 08:05 130848 /lib/ld-2.10.1.so
b773b000-b773c000 r--p 0001b000 08:05 130848 /lib/ld-2.10.1.so
b773c000-b773d000 rw-p 0001c000 08:05 130848 /lib/ld-2.10.1.so
bfd28000-bfd3e000 rw-p 00000000 00:00 0 [stack]
ffffe000-fffff000 r-xp 00000000 00:00 0 [vdso]
```

come si può vedere degli indirizzi dei segmenti, il kernel alloca memoria una pagina alla volta (4k ossia 0x1000); quindi l'indirizzo dei segmenti è allineato a 4k.

Lo stack è ad indirizzi alti; lo spazio degli indirizzi in un sistema a 32 bit è diviso in due blocchi: user space (0x00000000 – 0xc0000000) e kernel space (oltre 0xc0000000). Lo stack si trova vicino al limite di user space ed è assegnato in maniera pseudo-casuale se è abilitato ASLR (address space layout randomization).

## CHIAMATA AD UNA FUNZIONE DI LIBRERIA

### Procedura di “lazy binding” tramite Procedure Linkage Table.

I programmi che usano librerie condivise contengono chiamate a molte funzioni. Durante l'esecuzione, ci sono parti di codice che non vengono eseguite (routine di gestione errori, funzioni legate a menù di selezione non selezionati, ecc.). Quindi diverse funzioni non vengono mai chiamate. Perciò, per velocizzare la partenza del programma, i riferimenti alle funzioni vengono risolti solo all'atto della chiamata della funzione stessa. ELF sfrutta questo comportamento di rinvio della risoluzione dei riferimenti, detto “lazy binding”, attraverso la Procedure Linkage Table (PLT). Ogni programma con linking dinamico ad oggetti condivisi ed ogni libreria condivisa ha una PLT che contiene una voce per ogni routine non locale chiamata dal programma (o dalla libreria).

Nel nostro caso, la chiamata alla printf è questa (estratto dal disassemblato di .main):

```
8048405: e8 e6 fe ff ff call 80482f0 <printf@plt>
804840a: 8b 15 c8 96 04 08 mov edx,DWORD PTR ds:0x80496c8
```

Guardando nella tabella delle sezioni, si vede che l'indirizzo 80482f0 si trova nello spazio degli indirizzi

della .plt

```
[11] .plt          PROGBITS          080482c0 0002c0 000050 04 AX 0 0 4
[12] .text         PROGBITS          08048310 000310 0001cc 00 AX 0 0 16
```

Disassembiamo la sezione .plt

### \$ readelf -x 11 prova

Hex dump of section '.plt':

NOTE: This section has relocations against it, but these have NOT been applied to this dump.

```
0x080482c0 ff359c96 0408ff25 a0960408 00000000 .5.....%.....
0x080482d0 ff25a496 04086800 000000e9 e0ffffff .%.....h.....
0x080482e0 ff25a896 04086808 000000e9 d0ffffff .%.....h.....
0x080482f0 ff25ac96 04086810 000000e9 c0ffffff .%.....h.....
0x08048300 ff25b096 04086818 000000e9 b0ffffff .%.....h.....
```

o in modo più chiaro

### \$ objdump -d -j .plt prova

prova: file format elf32-i386

Disassembly of section .plt:

```
080482c0 <__gmon_start__@plt-0x10>:
80482c0: ff 35 9c 96 04 08      pushl 0x804969c
80482c6: ff 25 a0 96 04 08      jmp *0x80496a0
80482cc: 00 00                  add %al, (%eax)
...

080482d0 <__gmon_start__@plt>:
80482d0: ff 25 a4 96 04 08      jmp *0x80496a4
80482d6: 68 00 00 00 00         push $0x0
80482db: e9 e0 ff ff ff        jmp 80482c0 <_init+0x30>

080482e0 <__libc_start_main@plt>:
80482e0: ff 25 a8 96 04 08      jmp *0x80496a8
80482e6: 68 08 00 00 00         push $0x8
80482eb: e9 d0 ff ff ff        jmp 80482c0 <_init+0x30>

080482f0 <printf@plt>:
80482f0: ff 25 ac 96 04 08      jmp *0x80496ac
80482f6: 68 10 00 00 00         push $0x10
80482fb: e9 c0 ff ff ff        jmp 80482c0 <_init+0x30>

08048300 <puts@plt>:
8048300: ff 25 b0 96 04 08      jmp *0x80496b0
8048306: 68 18 00 00 00         push $0x18
804830b: e9 b0 ff ff ff        jmp 80482c0 <_init+0x30>
```

All'indirizzo che ci interessa 0x080482f0 c'è un salto indiretto (è il significato dell'asterisco premesso all'indirizzo) a 0x80496ac che appartiene alla .got.plt (la Global Offset Table caratteristica dei sistemi Intel)

```
[22] .got.plt       PROGBITS          08049698 000698 00001c 04 WA 0 0 4
```

```
[23] .data          PROGBITS          080496b4 0006b4 00000c 00  WA  0  0  4
```

Disassembiamo la GOT:

```
$ objdump -d -j .got.plt prova
```

```
prova:      file format elf32-i386
```

Disassembly of section .got.plt:

```
08049698 <_GLOBAL_OFFSET_TABLE_>:
8049698:  cc 95 04 08 00 00 00 00 00 00 00 d6 82 04 08
80496a8:  e6 82 04 08 f6 82 04 08 06 83 04 08
```

La locazione della GOT chiamata dal jmp indiretto contiene il valore 0x080482f6 (considerando l'orientamento little-endian). Questo non è altro che l'indirizzo dell'istruzione successiva della PLT. Quindi la prima chiamata alla GOT non fa nulla se non ritornare alla PLT

```
080482f0 <printf@plt>:
80482f0:  ff 25 ac 96 04 08      jmp     *0x80496ac
80482f6:  68 10 00 00 00      push  $0x10
80482fb:  e9 c0 ff ff ff      jmp     80482c0 <_init+0x30>
```

Questa istruzione inserisce nello stack, sopra l'indirizzo di ritorno salvato dalla chiamata della printf (0x804840a), un offset di rilocazione. Questo valore verrà utilizzato in seguito dal loader, e si rimanda a dopo la spiegazione.

Tornando alla PLT, l'istruzione successiva esegue il jmp alla prima voce della PLT, che ha etichetta PLT0:

```
080482f0 <printf@plt>:
80482f0:  ff 25 ac 96 04 08      jmp     *0x80496ac
80482f6:  68 10 00 00 00      push  $0x10
80482fb:  e9 c0 ff ff ff      jmp     80482c0 <_init+0x30>
```

L'istruzione nella PLT0 è la seguente:

```
080482c0 <__gmon_start__@plt-0x10>:
80482c0:  ff 35 9c 96 04 08      pushl  0x804969c
80482c6:  ff 25 a0 96 04 08      jmp     *0x80496a0
80482cc:  00 00                  add    %al, (%eax)
```

Essa inserisce nello stack la seconda voce della GOT (indirizzo **0x804969c**). Poi si salta alla istruzione successiva nella GOT all'indirizzo **0x80496a0**.

```
08049698 <_GLOBAL_OFFSET_TABLE_>:
8049698:  cc 95 04 08 00 00 00 00 00 00 00 d6 82 04 08
80496a8:  e6 82 04 08 f6 82 04 08 06 83 04 08
```

Sia questa locazione che la precedente contengono 0 a compile time. Esse vengono riempite a runtime. Nella locazione all'indirizzo **0x804969c** viene inserito un numero che identifica la libreria a cui appartiene la funzione (in questo caso la libc), in quella all'indirizzo **0x80496a0** viene inserito l'indirizzo della routine del loader per la risoluzione dei simboli.

A questo punto lo stack contiene nell'ordine l'indirizzo di ritorno della printf, l'offset della voce di rilocazione per printf e il nome della libreria da utilizzare (il loader), ed il controllo del programma è stato passato alla routine di risoluzione dei simboli del loader. Essa estrae dallo stack per primo il valore di identificazione della libreria, poi l'offset di rilocazione (0x10) e procede a eseguire la rilocazione della funzione printf.

Questa è la stampa delle voci di rilocazione dinamica

```
Contents of section .rel.plt:
 8048270 a4960408 07010000 a8960408 07020000 .....
 8048280 ac960408 07030000 b0960408 07040000 .....
```

L'offset 0x10 punta al campo evidenziato. Una voce di rilocazione ha il seguente formato:

```
typedef struct {
    Elf32_Addr r_offset;    /* Address */
    Elf32_Word r_info;     /* Relocation type and symbol index */
} Elf32_Rel;
```

Il primo campo è un indirizzo, ed in questo caso vale 0x080496ac, che è nello spazio della GOT e contiene, per ora, un rinvio alla PLT, ma dopo l'elaborazione della rilocazione conterrà l'indirizzo assoluto di printf. Il campo info si interpreta con le macro seguenti

```
#define ELF32_R_SYM(val) ((val) >> 8)
#define ELF32_R_TYPE(val) ((val) & 0xff)
#define ELF32_R_INFO(sym, type) (((sym) << 8) + ((type) & 0xff))
```

Il valore in questo caso è 0x0307 che significa: simbolo = 3 e tipo = 7.

Il tipo 7 corrisponde alla rilocazione R\_386\_JMP\_SLOT e il simbolo 3 si riferisce a printf nella tabella dinamica dei simboli, che possiamo stampare con il comando seguente:

### \$ readelf -s prova

Symbol table '.dynsym' contains 6 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	00000000	0	NOTYPE	WEAK	DEFAULT	UND	__gmon_start__
2:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	__libc_start_main@GLIBC_2.0 (2)
<b>3:</b>	<b>00000000</b>	<b>0</b>	<b>FUNC</b>	<b>GLOBAL</b>	<b>DEFAULT</b>	<b>UND</b>	<b>printf@GLIBC_2.0 (2)</b>
4:	00000000	0	FUNC	GLOBAL	DEFAULT	UND	puts@GLIBC_2.0 (2)
5:	080484fc	4	OBJECT	GLOBAL	DEFAULT	14	_IO_stdin_used

Lo stesso risultato si ottiene con il comando:

### \$ readelf -r prova

Relocation section '.rel.dyn' at offset 0x268 contains 1 entries:

Offset	Info	Type	Sym.Value	Sym. Name
08049694	00000106	R_386_GLOB_DAT	00000000	__gmon_start__

Relocation section '.rel.plt' at offset 0x270 contains 4 entries:

Offset	Info	Type	Sym.Value	Sym. Name
080496a4	00000107	R_386_JUMP_SLOT	00000000	__gmon_start__
080496a8	00000207	R_386_JUMP_SLOT	00000000	__libc_start_main
<b>080496ac</b>	<b>00000307</b>	<b>R_386_JUMP_SLOT</b>	<b>00000000</b>	<b>printf</b>
080496b0	00000407	R_386_JUMP_SLOT	00000000	puts

Il loader salva i registri e chiama una routine al suo interno per eseguire la rilocazione con i dati in suo possesso e infine inserisce nella GOT l'indirizzo reale della printf. Fatto ciò, ripristina il valore dei registri e trasferisce il controllo alla destinazione estraendo dallo stack l'indirizzo di ritorno della printf.

Nella eventuale successiva chiamata alla printf, l'esecuzione della voce nella PLT trasferisce il controllo alla GOT che ora contiene l'indirizzo di printf. La funzione viene quindi invocata direttamente senza passare attraverso il loader.

## **Versioni delle librerie**

Le librerie condivise hanno numeri di versione maggiore e minore (es. libc.so.1.1), ma i programmi sono collegati solo tramite il numero maggiore (es. libc.so.1), dato che si suppone che il numero minore sia retrocompatibile.

Per velocizzare il caricamento dei programmi, il sistema mantiene una cache con il pathname della versione più recente di ogni libreria, aggiornata ogni volta che si installa una nuova versione.

Ogni libreria ha un nome, il SONAME, assegnato all'atto della sua creazione. Ad es. libc.so.1.1 ha SONAME libc.so.1. In fase di linking il linker elenca il SONAME di tutte le librerie che collega. Il programma che crea la cache cerca tutte le librerie nell'albero delle directory delle librerie condivise, estrae il SONAME di ognuna di esse e, se ci sono omonimie, considera solo la versione più recente (numero di versione più alto). Poi aggiorna la cache coi SONAME e i pathname completi, in modo che a runtime il loader possa trovare velocemente la versione corrente di ogni libreria.

05 dicembre 2011

Per osservazioni: [and.cassani@gmail.com](mailto:and.cassani@gmail.com)