用GDB觀察共享庫函數的翻譯過程

研究了一下共享庫函數是怎樣加載到當前進程中的.開始共享庫函數地址放在GOT中,
第一次調用時,ld將其翻譯成函數在程序空間的真實地址.用GDB跟蹤了一下整個過程,
記錄在下面.

PLT (Procedure Linkage Table) 和 GOT (Global Offset Table)背景,google.

準備 ### ### 環境Ubuntu 11.04 amd64, ### 安裝libc debug symbol.

```
sudo apt-get install libc-dbg
```

安裝libc6 source，假設目錄是~/codes/debsrc/eglibc-2.13

```
sudo apt-get install build-essential
sudo apt-get source libc6
```

使用實驗源文件http://files.cnblogs.com/dyno/plt.zip,

```
mkdir whatever; cd whatever; make
main.c  <---- 主程序
test.c  <---- 共享庫
test.h
Makefile
```

foo & foo2是兩個共享庫中的函數，

```
[dyno@ubuntu:plt]$ objdump --syms main.exe | grep -E "(foo|xyz)"
0000000000000000      F *UND*  0000000000000000              foo2
<---- 1
0000000000000000      F *UND*  0000000000000000              foo
<---- 2
0000000000601028 g    O .bss   0000000000000004              xyz

[dyno@ubuntu:plt]$ readelf --sections --wide main.exe | grep got
  [22] .got              PROGBITS        0000000000600fe0 000fe0
000008 08  WA  0   0  8
  [23] .got.plt          PROGBITS        0000000000600fe8 000fe8
000030 08  WA  0   0  8
```

實驗

```
export LD_LIBRARY_PATH=$PWD
gdb main.exe

(gdb) break main
(gdb) run
Breakpoint 1, main () at main.c:4
4     xyz = 100;
```

加載ld的符號表，(/usr/lib/debug/lib/*是libc6-dbg安裝的debug symbol。)
注意 add-symbol-file的第三個參數，地址是如何得到的。

```
(gdb) info sharedlibrary
From                To                  Syms Read   Shared
Object Library
0x00007ffff7ddcaf0  0x00007ffff7df5a66  Yes (*)
/lib64/ld-linux-x86-64.so.2
0x00007ffff7bda500  0x00007ffff7bda628  Yes
/home/dyno/codes/plt/libtest.so
0x00007ffff7864c00  0x00007ffff79817ec  Yes
/lib/x86_64-linux-gnu/libc.so.6

(gdb) add-symbol-file
/usr/lib/debug/lib/x86_64-linux-gnu/ld-2.13.so 0x00007ffff7ddcaf0
(gdb) directory ~/codes/debsrc/eglibc-2.13/elf
(gdb) set disassemble-next-line on
```

foo() 現在是 <foo@plt>

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000400674 <+0>: push   %rbp
   0x0000000000400675 <+1>: mov      %rsp,%rbp
   0x0000000000400678 <+4>: movl   $0x64,0x2009a6(%rip)         #
0x601028 <xyz>
   0x0000000000400682 <+14>:  mov    $0x0,%eax
   0x0000000000400687 <+19>:  callq  0x400578 <foo@plt> <----
   0x000000000040068c <+24>:  mov    $0x0,%eax
   0x0000000000400691 <+29>:  callq  0x400578 <foo@plt>
...

(gdb) disassemble 0x400578
Dump of assembler code for function foo@plt:
   0x0000000000400578 <+0>: jmpq   *0x200a92(%rip)         #
0x601010 <_GLOBAL_OFFSET_TABLE_+40>
   0x000000000040057e <+6>: pushq  $0x2              <----
   0x0000000000400583 <+11>:  jmpq   0x400548  <----
End of assembler dump.
</foo@plt></foo@plt></xyz>
```

pushq是什麼？翻譯函數所需要的參數，這個是第一個參數reloc_index，是函數foo在GOT中的偏移量。
$rip裡存了下一條指令,所以實際上將要執行順序下一條指令

```
(gdb) p/x 0x40057e + 0x200a92
$3 = 0x601010
```

這就是PLT的精妙之處，第一次執行，轉到哪裡去了呢？

```
(gdb) disassemble 0x400548
```

```
No function contains specified address.
(gdb) x/5i 0x400548
   0x400548:    pushq  0x200aa2(%rip)      # 0x600ff0
<_GLOBAL_OFFSET_TABLE_+8>  <----
   0x40054e:    jmpq   *0x200aa4(%rip)     # 0x600ff8
<_GLOBAL_OFFSET_TABLE_+16> <----
   0x400554:    nopl   0x0(%rax)
   0x400558 <__libc_start_main@plt>:  jmpq   *0x200aa2(%rip)
     # 0x601000 <_GLOBAL_OFFSET_TABLE_+24>
   0x40055e <__libc_start_main@plt+6>:     pushq  $0x0
```
又一個pushq， link_map .got.plt,是翻譯需要的第二個參數。
再次jumpq，where？where？

```
(gdb) x/a 0x600ff8
0x600ff8 <_GLOBAL_OFFSET_TABLE_+16>:  0x7ffff7df0760
(gdb) info symbol 0x7ffff7df0760
_dl_runtime_resolve in section .text of
/usr/lib/debug/lib/x86_64-linux-gnu/ld-2.13.so
```
看看_dl_runtime_resolve是怎麼工作的...
```
(gdb) break _dl_runtime_resolve
(gdb) info breakpoints
Num   Type           Disp Enb Address            What
1     breakpoint     keep y   0x0000000000400678 in main at
main.c:4
    breakpoint already hit 1 time
2     breakpoint     keep y   0x00007ffff7df0760
../sysdeps/x86_64/dl-trampoline.S:30

(gdb) si
0x0000000000400548 in ?? ()
=> 0x0000000000400548:     ff 35 a2 0a 20 00  pushq  0x200aa2(%rip)
     # 0x600ff0 <_GLOBAL_OFFSET_TABLE_+8>
```
上面提到的第二個參數
```
(gdb) x/x 0x600ff0
0x600ff0 <_GLOBAL_OFFSET_TABLE_+8>: 0x00007ffff7ffe2e8

(gdb) list _dl_runtime_resolve
...
29  _dl_runtime_resolve:
30  subq $56,%rsp
31  cfi_adjust_cfa_offset(72) # Incorporate PLT
32  movq %rax,(%rsp)  # Preserve registers otherwise clobbered.
...
(gdb) list +
...
39  movq 64(%rsp), %rsi # Copy args pushed by PLT in register.
```

```
40  movq 56(%rsp), %rdi # %rdi: link_map, %rsi: reloc_index  <----
前面提到的兩個參數
41  call _dl_fixup    # Call resolver.
42  movq %rax, %r11   # Save return value        <----真正的共享庫裡函數
地址
43  movq 48(%rsp), %r9  # Get register content back.
...
```

設置斷點，看地址在GOT表中的變化

```
(gdb) info line _dl_runtime_resolve
Line 30 of "../sysdeps/x86_64/dl-trampoline.S" starts at address
0x7ffff7df0760 <_dl_runtime_resolve>
   and ends at 0x7ffff7df0764 <_dl_runtime_resolve+4>.
(gdb) break ../sysdeps/x86_64/dl-trampoline.S:40
Breakpoint 3 at 0x7ffff7df078b: file
../sysdeps/x86_64/dl-trampoline.S, line 40.

(gdb) c
(gdb) x/a 0x601010
0x601010 <_GLOBAL_OFFSET_TABLE_+40>:  0x40057e <foo@plt+6>  <----
_dl_fixup 之前
(gdb) ni
42  movq %rax, %r11   # Save return value
=> 0x00007ffff7df0795 <_dl_runtime_resolve+53>:      49 89 c3   mov
     %rax,%r11
(gdb) x/a 0x601010
0x601010 <_GLOBAL_OFFSET_TABLE_+40>:  0x7ffff7bda5cc <foo>  <----
_dl_fixup 之後
</foo></foo@plt+6>
```

以後再次調用foo就直接到這裡了。

延伸閱讀

[1] Reversing the ELF Stepping with GDB during PLT uses and .GOT fixup
    http://packetstormsecurity.org/files/view/25642/elf-runtime-fixup.txt
[2] AMD64 Application Binary Interface (v 0.99)
    http://www.x86-64.org/documentation/abi.pdf
[3] PLT and GOT - the key to code sharing and dynamic libraries
    http://www.technovelty.org/linux/pltgot.html
[4] examining PLT/GOT structures
    http://althing.cs.dartmouth.edu/secref/resources/plt-got.txt
[5] Debugging with GDB
    http://sourceware.org/gdb/current/onlinedocs/gdb/
[6] 共享庫函數調用原理
    http://blog.csdn.net/absurd/article/details/3169860
[7] How main() is executed on Linux
    http://linuxgazette.net/issue84/hawk.html

[8] Gentle Introduction to x86-64 Assembly
http://www.x86-64.org/documentation/assembly.html