

延迟绑定

这是一个简化的教程

接下来的内容是以下面这段代码为例：

```
#include<stdio.h>
void main(){
    puts("Hello World!");
}
```

首先我们先看一下这个程序的GOT表中的内容：

```
objdump -R hello
```

```
# liqingyuan @ ubuntu in ~/Desktop [15:39:09]
$ objdump -R hello

hello:      file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET      TYPE          VALUE
08049ffc    R_386_GLOB_DAT  __gmon_start__
0804a00c    R_386_JUMP_SLOT puts
0804a010    R_386_JUMP_SLOT __gmon_start__
0804a014    R_386_JUMP_SLOT __libc_start_main
```

我们可以看到 puts 对应的 地址为 0x0804a00c

现在我们看一下 在执行到puts函数之前 这个0x0804a00c地址中的内容是什么。

```

gdb-peda$ disassemble main
Dump of assembler code for function main:
0x0804841d <+0>:    push    ebp
0x0804841e <+1>:    mov     ebp,esp
0x08048420 <+3>:    and     esp,0xffffffff
0x08048423 <+6>:    sub     esp,0x10
0x08048426 <+9>:    mov     DWORD PTR [esp],0x80484d0
0x0804842d <+16>:   call    0x80482f0 <puts@plt>
0x08048432 <+21>:   leave
0x08048433 <+22>:   ret
End of assembler dump.

```

我们在调用puts函数前 下一个断点 `b * 0x0804842d` 执行

此时我们看一下 刚才GOT表中 地址 0x0804a00c的内容是什么。

```
x/wx 0x0804a00c
```

```

gdb-peda$ x/wx 0x0804a00c
0x0804a00c <puts@got.plt>:      0x080482f6
gdb-peda$

```

然后 我们 在执行完puts函数之后再看一下 这个地址中的内容

```

gdb-peda$ x/wx 0x0804a00c
0x0804a00c <puts@got.plt>:      0xb7e797e0
gdb-peda$

```

由上面两张图片可知 GOT表中的地址中的内容发生了变化

那变化后的内容是什么呢?

我们现在 反汇编一下puts函数

```
gdb-peda$ disassemble puts
Dump of assembler code for function _IO_puts:
   0xb7e797e0 <+0>:      push    ebp
   0xb7e797e1 <+1>:      push    edi
   0xb7e797e2 <+2>:      push    esi
   0xb7e797e3 <+3>:      push    ebx
   0xb7e797e4 <+4>:      sub     esp,0x1c
   0xb7e797e7 <+7>:      call   0xb7f3cc6b <__x86.get_pc_thunk.bx>
   0xb7e797ec <+12>:     add     ebx,0x147814
   0xb7e797f2 <+18>:     mov     eax,DWORD PTR [esp+0x30]
   0xb7e797f6 <+22>:     mov     DWORD PTR [esp],eax
```

由此我们看到 变化后的内容就是 puts函数的起始地址。

这就是直观的延迟绑定。

用于快速回忆 延迟绑定 的内容。

详细内容可以参看 长亭系统安全ppt第一章。