栈迁移

栈迁移主要是为了解决栈溢出可以溢出空间大小不足的问题，或者是因为ASLR栈地址不可预测等用于制造一个全新可控的空间位置。

基础

要想明白栈迁移起码得知道以下的汇编语句的含义是什么。

```
等价汇编语句一：
leave
ret
————————
mov ebp,esp
pop ebp
```

```
等价汇编语句二：
ret
————————
pop
```

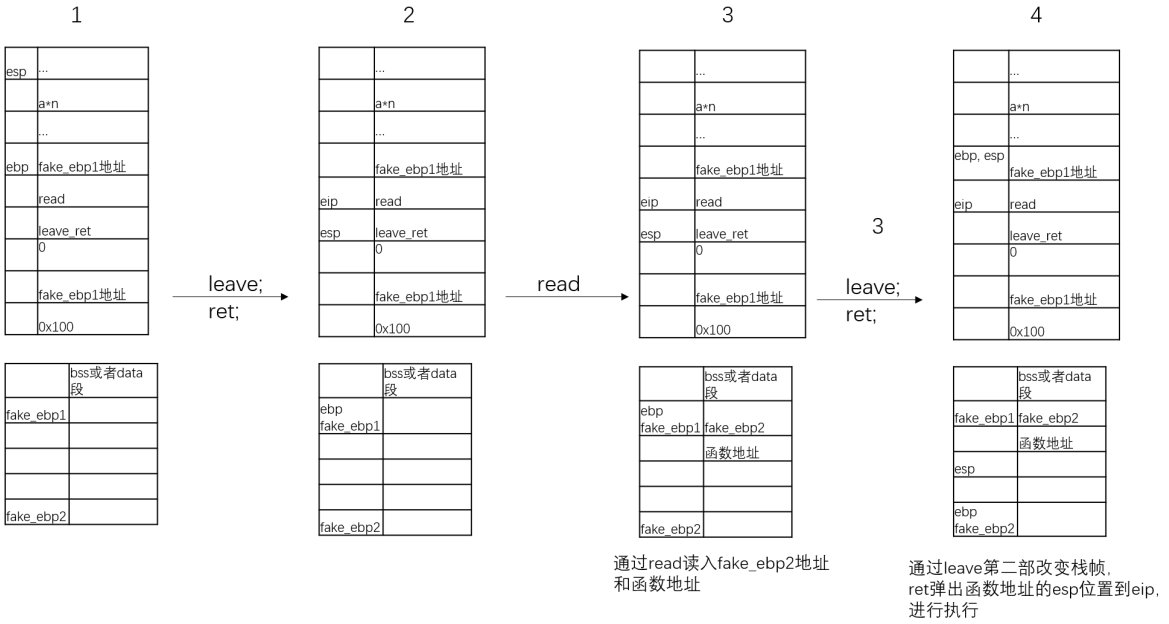
```
等价汇编语句三：
call 标号
————————
push IP
jmp near ptr 标号
```

原理概述

栈迁移主要是利用leave；ret；语句进行操作。很好理解，这个语句可以进行对栈指针的修改操作，并且还可以对eip进行移动，使得程序顺利的迁移。

操作体验

多说无益，为了讲解清晰，最好的办法是自己走一遍，就能很快明白这个迁移是怎么个搞法。



1.在状态1中，展示了其栈帧的排布示意图。当程序完成call执行之后，会有leave；ret的操作，此时进行第一次栈帧的变化。肢解其步骤：

- mov esp,ebp。将esp移动到ebp的位置。
- pop ebp。将esp上对应的内容赋给ebp(fake_ebp1)，ebp转移到bss中，同时esp需要减一个单位。
- pop eip。将此时esp对应位置上的地址给eip（read的地址），esp继续减一。

2.上述步骤完成到达状态2.此时执行eip上对应地址对应的函数（read），我们可以进行读取操作，此时读入fake_ebp2以及下一步需要执行的函数地址。

3.执行上述操作后栈的排布如状态图3，执行完read函数之后，会有ret语句，此时我们会再次执行leave；ret；肢解其步骤：

- mov esp,ebp。将esp移动到ebp的位置，esp转移到bss中。
- pop ebp。将esp上对应的内容赋给ebp(fake_ebp2),ebp向下大部分拉动变化，同时esp需要减一个单位。
- pop eip。将此时esp对应位置上的地址给eip（函数地址），esp继续减一。

4.之后边执行eip中被给的地址上对应的函数，而栈就被迁移到我们设置的位置中了。

多次思维练习

跟着过完操作步骤并且理解以后，其实已经完成了栈迁移的学习，但是为了让后续遇到栈迁移不至于卡顿建议进行进一步思维“深造”。

buf（ebp）	//pop数值到ebp改变基址
read(ret)	//pop到eip，执行read(0,buf,0x100)
ret	//执行完read后返回，pop到eip
0	
buf	
0x100	

执行流不中断

buf	//pop数值到ebp改变基址
read	//pop到eip，执行read(0,buf,0x100)
leave_ret	//pop到eip,执行完后esp转移了
0	//栈指针都已经转移了
buf	
0x100	

上述为初始栈上的位置的排布情况，下图为bss段构造的思路。

泄露地址		
./bss	buf2	//pop ebp改变基址
	puts_plt	//装载用于打印泄露的puts
	pop_ret	//puts之后，pop到eip执行
	puts_gots	//要泄露的参数值
	read_plt	//pop到eip执行，执行read(0，buf，0x00)
	leave_ret	//pop到eip执行，执行leave_ret，esp迁移
	0	
	buf	
	0x100	

调用system函数		
./bss	bbbb	//随便，改变ebp基址
	system	//leave_ret，ret后调用
	cccc	//返回地址，可随意
	bin_sh_addr	

> 参考博文：
>
> https://cloud.tencent.com/developer/article/1601192
>
> https://www.cnblogs.com/zuoanfengxi/p/12808171.html