# ROP原理：

ROP 不调用系统库中已有的函数，而是把程序中已有的以Ret 结尾的短指令片段（一般不超过5 条指令）进行再组合,这些被利用的指令片段被称为Gadget。ROP 攻击通过溢出， 将Gadget 的地址预先注入到栈中,然后劫持控制流,将程序执行流转向第一个ROP Gadget, 当第一个ROP Gadget执行完毕后,利用Ret 指令的特性,跳转到栈顶所指向的下一个Gadget。这样依次重复,可以将多个Gadget 串联执行,以实现特定的功能。

# 分析过程：

x86中参数都是保存在栈上,但在x64中前六个参数依次保存在RDI, RSI, RDX, RCX, R8和 R9寄存器里，如果还有更多的参数的才会保存在栈上。

利用ROP攻击执行系统调用execv(“/bin/sh”)，系统调用号为0x3b，将系统调用号保存到寄存器rax中，其中execv()所需的参数”/bin/sh”保存在rdi寄存器中，rdx和rsi中置0，。

利用ROPGadget在/lib/x86\_64-linux-gnu/libc.so.6中搜素所需gadget从而获得gadget的地址。

构造payload：首先找到漏洞程序中返回地址rip距离buf的偏移(40)，

然后确定gadget的位置以及参数”/bin/sh”的位置。

payload = "a"\*offset + ret1\_addr + binsh\_addr + ret2\_addr + ret3\_addr + data1 + ret4\_addr + data2 + data3 + ret5\_addr

# 所用gadget：

ret1：pop %rdi; ret

ret2：xor %rax %rax; ret

ret3：pop %rax; ret

ret4：pop %rdx; pop %rsi; ret

ret5：syscall

# 栈：

Payload在栈中的情况：

|  |
| --- |
| a  .  .  . |
| ret1 |
| The address of string ‘/bin/sh’ |
| ret2  地址递增 |
| ret3 |
| 0x3b execv syscall number |
| ret4 |
| 0 |
| 0 |
| ret5 |

# 32位ROP攻击分析：

1. 当系统调用参数<=5个时:   
    eax中存放系统调用的功能号，传递给系统调用的参数顺序依次放到寄存器:ebx,ecx,edx,esi,edi中
2. 当系统调用参数>5个时:   
    eax中存放系统调用的功能号，全部参数应依次放在一块连续的内存区域中，同时在寄存器ebx中保存指向该内存区域的指针(内存块的首地址)；linux采用的是C调用模式，这意味着如果要以stack来实现这个连续的内存块（当然可以用其他方式实现，比如heap）时，所有参数必须以相反的顺序进栈：即最后一个参数最先进栈，第一个参数最后进栈，最后将栈指针的值（栈顶地址）放到寄存器ebx中。

# 所用gadget：

ret1：pop %ebx; ret

ret2：xor %eax %eax; ret

ret3：pop %eax; ret

ret4：pop %ecx; pop %edx; ret

ret5：int 80

# static所用gadget：

ret1：pop %ebx; ret

ret2：xor %eax %eax; ret

ret3：pop %eax; ret

ret4：pop %ecx; ret

ret5: pop %edx; ret

ret6：int 80