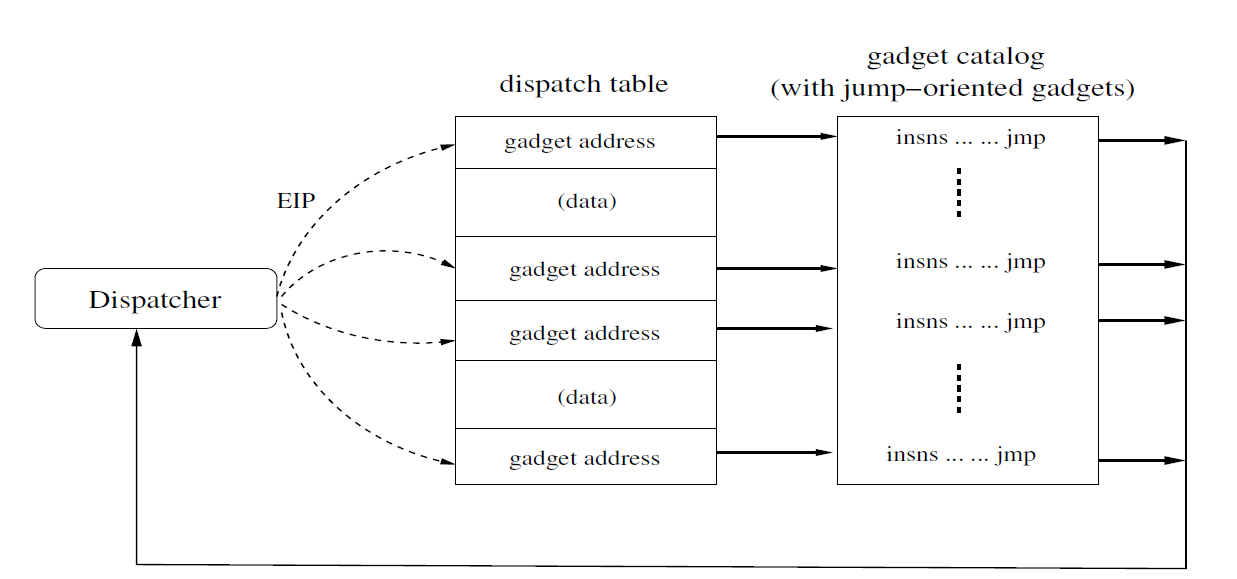
# JOP原理：

In a JOP-based attack, the attacker abandons all reliance on the stack for control ow and ret for gadget discovery and chaining, instead using nothing more than a sequence of indirect jump instructions.

Similar to ROP, the building blocks of JOP are still short code sequences called gadgets. However, instead of ending with a ret, each such gadget ends with an indirect jmp.

The dispatcher gadget is intended to govern control flow among various jump-oriented gadgets. Functional gadgets that perform primitive operations, this

dispatcher gadget is specically selected to determine which functional gadget is going to be invoked next. Naturally, the dispatcher gadget can maintain an internal dispatch table that explicitly species the control ow of the functional gadgets. Also, it ensures that the ending jmp instruction in the functional gadget will always transfer the control back to the dispatcher gadget. By doing so, jump-oriented computation

becomes feasible.

图1 The jop model

# 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JOP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。

由于jop gadget 不容易找到，所以在漏洞程序中嵌入了一小段汇编，方便攻击。

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，进而构造payload

# 所用gadget：

## dispatcher gadget：

L1: pop %rcx

pop %rax

pop %rbx

L2 : add $0x08,%rbx

jmpq \*(%rbx)

## functional gadget：

JMP1： pop %rdi; jmp %rax

JMP2： pop %rsi; pop %r15; jmp %rax

JMP3： pop %r12; jmp %rax

JMP4： mov %r12, %rdx; call %rax

JMP5： pop %rax; jmp %rcx

JMP6： pop %rax; jmp %rcx

JMP7： Syscall

# 栈：

Payload 在栈中的存放

|  |
| --- |
| L1 |
| L2 |
| L2 |
| the address of first functional gadget address in stack |
| The address of string ‘/bin/sh’  地址递增 |
| 0 |
| 0 |
| 0 |
| 0x3b execv syscall number |
| JMP1 |
| JMP2 |
| JMP3 |
| JMP4 |
| JMP5 |
| JMP6 |
| JMP7 |