

Enumeration and Exploitation

Binary 1 (Medium)

This challenge evaluates the contestant's ability to use a disassembler to exploit a compiled binary. One possible tool to use is the "GDB" Linux program. A Linux binary is provided and the contestant is tasked with extracting the secret flag. This can be solved by disassembling the main function, which reveals a call to "gets".

```
(gdb) disassemble main
Dump of assembler code for function main:
0x0000000040084e <+0>: push %rbp
0x0000000040084f <+1>: mov %rsp,%rbp
0x00000000400852 <+4>: sub $0x40,%rsp
0x00000000400856 <+8>: mov %edi,-0x34(%rbp)
0x00000000400859 <+11>: mov %rsi,-0x40(%rbp)
0x0000000040085d <+15>: cmpl $0x2,-0x34(%rbp)
0x00000000400861 <+19>: je 0x400886 <main+56>
0x00000000400863 <+21>: mov -0x40(%rbp),%rax
0x00000000400867 <+25>: mov (%rax),%rax
0x0000000040086a <+28>: mov %rax,%rsi
0x0000000040086d <+31>: mov $0x4009a2,%edi
0x00000000400872 <+36>: mov $0x0,%eax
0x00000000400877 <+41>: callq 0x400580 <printf@plt>
0x0000000040087c <+46>: mov $0x1,%edi
0x00000000400881 <+51>: callq 0x4005e0 <exit@plt>
0x00000000400886 <+56>: mov -0x40(%rbp),%rax
0x0000000040088a <+60>: add $0x8,%rax
0x0000000040088c <+64>: mov (%rax),%rax
0x00000000400891 <+67>: mov %rax,%rdi
0x00000000400894 <+70>: callq 0x400570 <strlen@plt>
0x00000000400899 <+75>: cmp $0x4,%rax
0x0000000040089d <+79>: je 0x4008c2 <main+116>
Type <return> to continue, or q <return> to quit
0x0000000040089f <+81>: mov -0x40(%rbp),%rax
0x000000004008a3 <+85>: mov (%rax),%rax
0x000000004008a6 <+88>: mov %rax,%rsi
0x000000004008a9 <+91>: mov $0x4009a2,%edi
0x000000004008ac <+96>: mov $0x0,%eax
0x000000004008b3 <+101>: callq 0x400580 <printf@plt>
0x000000004008b8 <+106>: mov $0x1,%edi
0x000000004008bd <+111>: callq 0x4005e0 <exit@plt>
0x000000004008c2 <+116>: movl $0x0,-0x4(%rbp)
0x000000004008c9 <+123>: mov $0x4009b3,%edi
0x000000004008ce <+128>: mov $0x0,%eax
0x000000004008d3 <+133>: callq 0x400580 <printf@plt>
0x000000004008d8 <+138>: lea -0x30(%rbp),%rax
0x000000004008dc <+142>: mov %rax,%rdi
0x000000004008df <+145>: callq 0x4005d0 <gets@plt>
0x000000004008e4 <+150>: cmpl $0x0,-0x4(%rbp)
0x000000004008e8 <+154>: je 0x4008f8 <main+170>
0x000000004008ea <+156>: mov -0x40(%rbp),%rax
0x000000004008ee <+160>: mov %rax,%rdi
0x000000004008f1 <+163>: callq 0x4005dd <fg>
0x000000004008f6 <+168>: jmp 0x400902 <main+180>
0x000000004008f9 <+170>: mov $0x4009cd,%edi
0x000000004008fd <+175>: callq 0x400560 <puts@plt>
Type <return> to continue, or q <return> to quit
0x00000000400902 <+180>: mov $0x0,%eax
0x00000000400907 <+185>: leaveq
0x00000000400908 <+186>: retq
End of assembler dump.
```

This is a function that is known to commonly have issues with buffer overflows. The disassembly also reveals the size of the buffer (30 bytes) which can be exploited if more than 30 bytes are inputted. Based on the disassembly, it can be inferred that the original code looks something along these lines:

```
char input[30];
int pass = 0;
printf("Please enter a password: ");
gets(input);
if (pass) {
    // Print out flag
} else {
    printf("Try again.\n");
}
```

Thus, the entering 31 bytes into the password prompt will overflow the buffer, overwriting the value of pass. Any ASCII character for the 31st character would make the pass variable a truthy value and print out the flag.

Question	Answer
What is the flag hidden in the program?	NCL-EZOF-5336