Лабораторная работа №7
«ИССЛЕДОВАНИЕ УЯЗВИМОСТИ «ПЕРЕПОЛНЕНИЕ БУФЕРА»

Цель работы: исследование уязвимости «переполнение буфера» на примере тестовой программы и операционной системы Ubuntu/Linux.

Порядок выполнения:
1. Скорректировать тестовый файл «main.c» в соответствии с вариантом задания.
2. Скомпилировать программу командой «gcc -O0 -mpreferred-stack-boundary=2 -g -m32 -fno-stack-protector main_var?.c»
3. Протестировать полученную программу путём ввода неправильного пароля, правильного пароля и неправильного пароля, длина которого превышает размер выделенного буфера.
4. Запустить отладчик **gdb**, задать точку останова на функцию **main()** и получить результат дизассемблирования функции **main()** и **buff_overflow_test()**.
5. При помощи пошаговой отладки программы в **gdb** посмотреть содержимое стека для буфера **buff_var** для случаев произвольного короткого пароля и пароля, длина которого превышает размер буфера на 1 байт. Привести результат выполнения программы при вводе этих паролей.
6. В качестве отчёта по лабораторной работе привести результаты выполнения указанных выше пунктов, в том числе скорректированный исходный код и результаты работы отладчика **gdb**.

Таблица 1 – Варианты заданий

№ бригады	**1**	**2**	**3**	**4**	**5**
Название исходного файла	main_var1	main_var2	main_var3	main_var4	main_var5
Название буфера	buff_var1	buff_var2	buff_var3	buff_var4	buff_var5
Размер буфера, байт	5	6	7	8	9
Правильный пароль	pass1	pass2	pass3	pass4	pass5

Таблица 1 – Варианты заданий (продолжение)

№ бригады	**6**	**7**	**8**	**9**	**10**
Название исходного файла	main_var6	main_var7	main_var8	main_var9	main_var10
Название буфера	buff_var6	buff_var7	buff_var8	buff_var9	buff_var10
Размер буфера, байт	10	11	12	13	14
Правильный пароль	pass6	pass7	pass8	pass9	pass10

Пример выполнения

Необходимое программное обеспечение:
1. Операционная система Ubuntu/Linux.
2. Установленные компилятор **gcc** и отладчик **gdb**.

3. Редактор текста с подсветкой синтаксиса **gedit** (или другой).

Вариант для примера:

- название исходного файла – main.c
- название буфера – buff_var0
- размер буфера, байт – 4
- правильный пароль – pasw

1. Исходный код тестовой программы (main.c).

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

int main(void)
{
  buff_overflow_test();
  return 0;
}

int buff_overflow_test()
{
  char buff_var0[4];                          ; исследуемый буфер
  int pass = 0;

  printf("\n Enter the password : \n");
  gets(buff_var0);                            ; ввод значений в буфер

  if(strcmp(buff_var0, "pasw"))               ; сравнение значения
  {
    printf ("\n Wrong Password \n");
  }
  else
  {
    printf ("\n Correct Password \n");
    pass = 1;
  }

  if(pass)                                    ; участок кода, выполняемый
  {                                           ; если пароль верный
   /* Now Give root or admin rights to user*/
    printf ("\n Root privileges given to the user \n");
  }
}
```

Команда для создания исполняемого файла с отключением механизмов защиты адресного пространства и оптимизаций:

gcc -O0 -mpreferred-stack-boundary=2 -g -m32 -fno-stack-protector main.c

Результат выполнения компилятора:

```
main.c: In function ‘main’:
main.c:7:5: warning: implicit declaration of function ‘buff_overflow_test’ [-Wimplicit-function-declaration]
    buff_overflow_test();
    ^
main.c: In function ‘buff_overflow_test’:
main.c:19:5: warning: implicit declaration of function ‘gets’ [-Wimplicit-function-declaration]
    gets(buff_var0);
    ^
/tmp/ccE51AAQ.o: In function `buff_overflow_test':
/laba_buff_overflow/src_lab/main.c:19: warning: the `gets' function is dangerous and should not be used.
```

2. Тестирование программы.

Запускаем исполняемый файл командой «./a.out» и вводим неверный пароль «pass»:

```
 Enter the password :
pass

 Wrong Password
```

Как видно, в случае ввода неправильного пароля программа отработала правильно и вывела сообщение о неправильном пароле.

Запускаем исполняемый файл командой «./a.out» и вводим верный пароль «pasw»:

```
 Enter the password :
pasw

 Correct Password

 Root privileges given to the user
```

В случае ввода правильного пароля программа отработала правильно и вывела сообщение о правильности пароля и о получении повышенных привилегий.

Запуск исполняемого файла командой «./a.out» и ввод неверного пароля «11111111111», который больше размера буфера:

```
 Enter the password :
11111111111

 Wrong Password

 Root privileges given to the user
```

В данном случае, программа выводит сообщение о том, что пароль неправильный, но были получены повышенные привилегии. Данный эффект достигается за счёт того, что введённые данные вышли за границы отведённого буфера и затёрли соседние области в стеке, в том числе переменную «**int pass**». Так как значение этой переменной отличается от «0», то происходит выполнение участка кода, дающего повышенные привилегии:

```
if(pass)                                ; участок кода, выполняемый
{                                       ; если пароль верный
 /* Now Give root or admin rights to user*/
  printf ("\n Root privileges given to the user \n");
}
```

3. Отладка программы.

Запускаем отладчик **gdb**:

```
 laba_buff_overflow/src_lab$ gdb a.out
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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:
<http://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 a.out...done.
(gdb)
```

Добавляем точку остановки на функцию **main** командой **break main** :

```
(gdb) break main
Breakpoint 1 at 0x804846e: file main.c, line 7.
```

Выводим дизассемблированный код для функции **main** и **buff_overflow_test** командами **disassemble main** и **disassemble buff_overflow_test** :

```
(gdb) disassemble main
```

```
Dump of assembler code for function main:
   0x0804846b <+0>:     push   %ebp
   0x0804846c <+1>:     mov    %esp,%ebp
   0x0804846e <+3>:     call   0x804847a <buff_overflow_test>
   0x08048473 <+8>:     mov    $0x0,%eax
   0x08048478 <+13>:    pop    %ebp
   0x08048479 <+14>:    ret
End of assembler dump.
(gdb) disassemble buff_overflow_test
Dump of assembler code for function buff_overflow_test:
   0x0804847a <+0>:     push   %ebp
   0x0804847b <+1>:     mov    %esp,%ebp
   0x0804847d <+3>:     sub    $0x8,%esp      ; выделение памяти в стеке для переменных
   0x08048480 <+6>:     movl   $0x0,-0x4(%ebp)
   0x08048487 <+13>:    push   $0x8048570
   0x0804848c <+18>:    call   0x8048340 <puts@plt> ; вызов функции printf()
   0x08048491 <+23>:    add    $0x4,%esp
   0x08048494 <+26>:    lea    -0x8(%ebp),%eax
   0x08048497 <+29>:    push   %eax
   0x08048498 <+30>:    call   0x8048330 <gets@plt> ; вызов функции gets()
   0x0804849d <+35>:    add    $0x4,%esp
   0x080484a0 <+38>:    push   $0x8048588
   0x080484a5 <+43>:    lea    -0x8(%ebp),%eax
   0x080484a8 <+46>:    push   %eax
   0x080484a9 <+47>:    call   0x8048320 <strcmp@plt> ; сравнение строк
   0x080484ae <+52>:    add    $0x8,%esp
   0x080484b1 <+55>:    test   %eax,%eax
   0x080484b3 <+57>:    je     0x80484c4 <buff_overflow_test+74>
   0x080484b5 <+59>:    push   $0x804858d
   0x080484ba <+64>:    call   0x8048340 <puts@plt>
   0x080484bf <+69>:    add    $0x4,%esp
   0x080484c2 <+72>:    jmp    0x80484d8 <buff_overflow_test+94>
   0x080484c4 <+74>:    push   $0x804859f
   0x080484c9 <+79>:    call   0x8048340 <puts@plt>
   0x080484ce <+84>:    add    $0x4,%esp
   0x080484d1 <+87>:    movl   $0x1,-0x4(%ebp)
   0x080484d8 <+94>:    cmpl   $0x0,-0x4(%ebp)
   0x080484dc <+98>:    je     0x80484eb <buff_overflow_test+113>
   0x080484de <+100>:   push   $0x80485b4
   0x080484e3 <+105>:   call   0x8048340 <puts@plt>
   0x080484e8 <+110>:   add    $0x4,%esp
   0x080484eb <+113>:   nop
   0x080484ec <+114>:   leave
   0x080484ed <+115>:   ret
End of assembler dump.
(gdb)
```

Обратим внимание на регистры указателей стека: ESP – текущий указатель стека, EBP – базовый указатель стека.
В строках

```
0x0804847a <+0>:     push   %ebp
0x0804847b <+1>:     mov    %esp,%ebp
```

сохраняется значение базового указателя стека и присваивание базовому указателю стека значения текущего указателя стека.

В строке

```
0x0804847d <+3>:     sub    $0x8,
```

показано изменение указателя стека на 8 байт командой **sub $0x8,%esp**. Таким образом в стеке выделяется 8 байт под переменные **buff_var0** и **pass** (4 байта для буфера и 4 байта для целочисленной переменной **pass**).

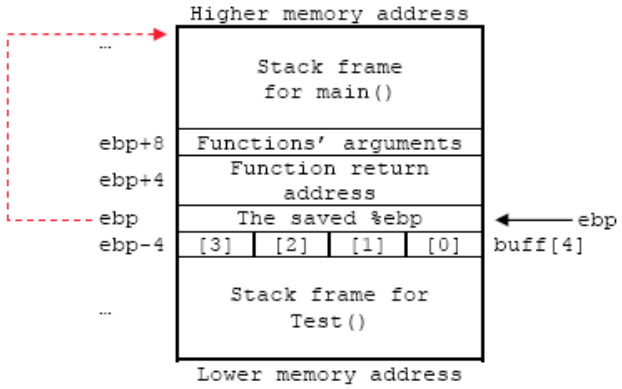

Рис. 1. Схема работы стека

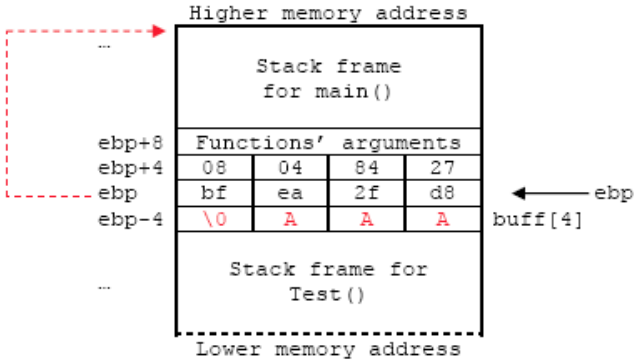

Рис. 2. Схема работы стека при вводе значения пароля «AAA»

Выйдем из отладчика **gdb** нажатием комбинации кнопок «CTRL-Z» и перезапустим его командой **gdb -q a.out**.

Установим точку остановки командой **break main**, после чего запустим программу командой **r** (run) и осуществим пошаговое выполнение программы командой **s** (step):

```
Reading symbols from a.out...done.
(gdb) break main
Breakpoint 1 at 0x804846e: file main.c, line 7.
(gdb) r
Starting program: /laba_buff_overflow/src_lab/a.out

Breakpoint 1, main () at main.c:7
7               buff_overflow_test();
(gdb) s
buff_overflow_test () at main.c:16
```

```
16          int pass = 0;
(gdb)
18          printf("\n Enter the password : \n");
(gdb)

 Enter the password :
19          gets(buff_var0);
```

В поле ввода введём пароль «AAA»

```
(gdb)
AAA
21          if(strcmp(buff_var0, "pasw"))
```

На 21 строке, в которой осуществляется сравнение введённого пароля с «pasw», посмотрим командами **x/x** и **x/s** содержимое памяти в стеке по адресу, указанному в регистре ESP:

```
(gdb) x/x $esp
0xffffcc98:     0x00414141
(gdb) x/s $esp
0xffffcc98:     "AAA"
```

Как видно, по заданному адресу хранится введённый нами пароль.
Посмотрим, что хранится в соседней области памяти. Мы знаем, что на наш буфер выделено 4 байта. Поэтому, прибавив значение 4 к значению указателя сможем посмотреть содержимое переменной «**int pass**»:

```
(gdb) x/s $esp+4
0xffffcc9c:     ""
(gdb) x/x $esp+4
0xffffcc9c:     0x00
(gdb)
```

Как видно, содержимое переменной «**int pass**» равно 0x00.
Перезапустим отладчик и повторим предыдущие шаги, только в качестве пароля введём значение «AAAAA», что на один байт больше, чем выделяется на буфер.

```
(gdb) x/x $esp
0xffffcc98:     0x41414141
(gdb) x/s $esp
0xffffcc98:     "AAAAA"
(gdb) x/x $esp+4
0xffffcc9c:     0x41
(gdb) x/s $esp+4
0xffffcc9c:     "A"
```

Как видно, содержимое переменной «**int pass**» равно 0x41. Если продолжить выполнение программы, то мы увидим, что пароль неправильный, но предоставлен привилегированный режим.

```
(gdb) s
23              printf ("\n Wrong Password \n");
(gdb)
```

```
 Wrong Password
31          if(pass)
(gdb)
34              printf ("\n Root privileges given to the user \n");
(gdb)

 Root privileges given to the user
36      }
(gdb)
main () at main.c:8
8           return 0;
(gdb)
```

Таким образом, успешно продемонстрирована возможность переполнения буфера для манипуляции значениями другой переменной.