

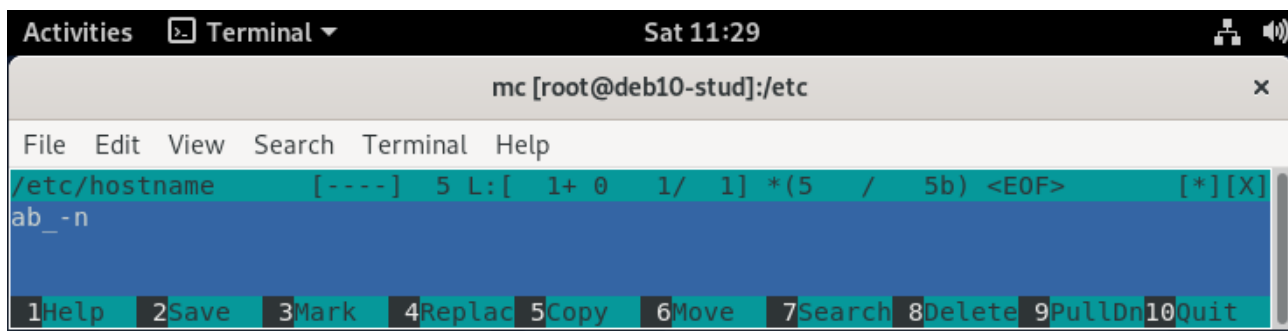
Расчетно-графическое задание

Освоение принципов разработки программных средств безопасности операционных систем и приложений.

Цель работы: изучить процесс выполнения кода программы на языке C в операционной системе Debian Linux и способы выявления уязвимостей.

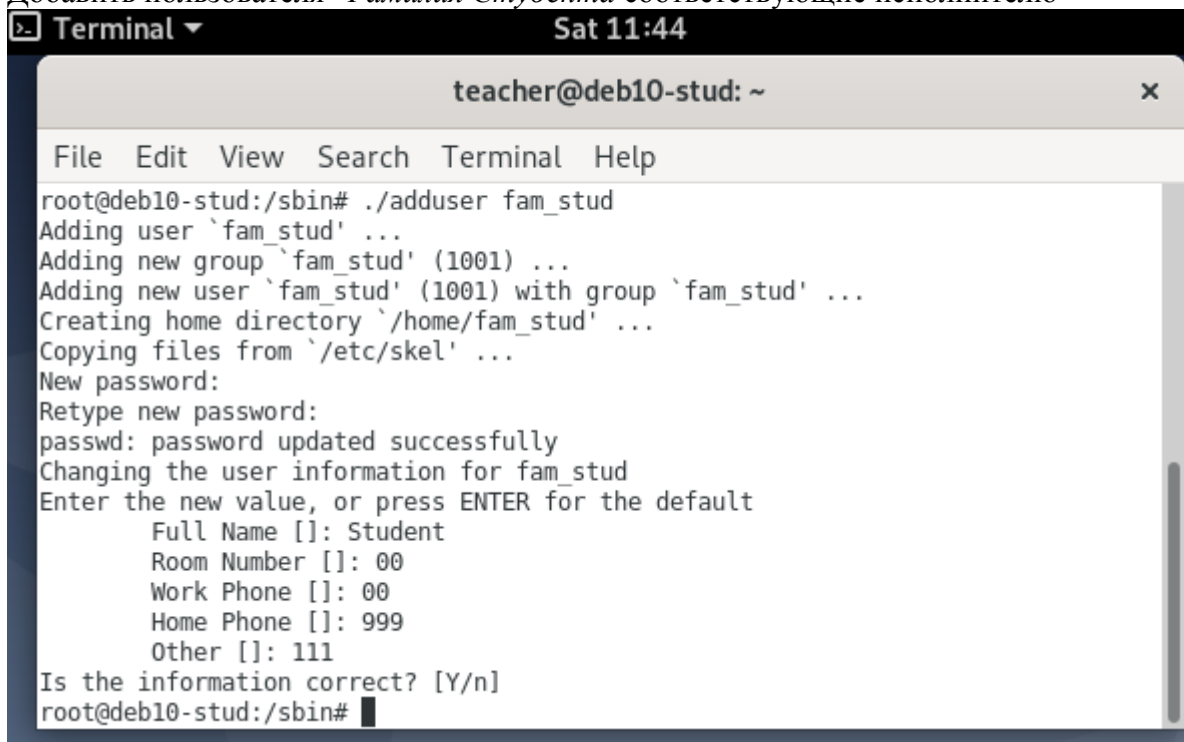
1. Настройка системы, установка и тестирование программного обеспечения.

Установить имя компьютера - *Группа_Номер по списку DiSpace Студента* соответствующие исполнителю..



```
mc [root@deb10-stud]:/etc
File Edit View Search Terminal Help
/etc/hostname [----] 5 L:[ 1+ 0 1/ 1] *(5 / 5b) <EOF> [*][X]
ab-n
1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn10Quit
```

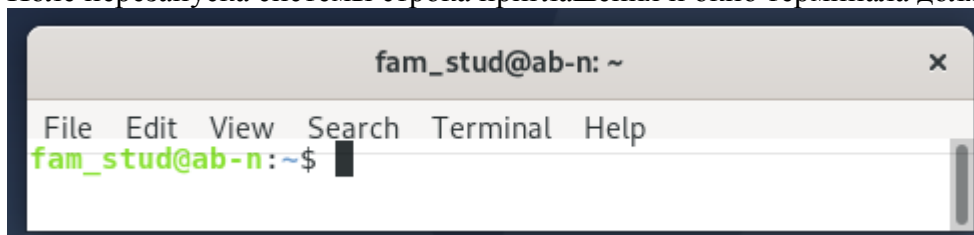
Добавить пользователя *Фамилия Студента* соответствующие исполнителю



```
teacher@deb10-stud: ~
File Edit View Search Terminal Help
root@deb10-stud:/sbin# ./adduser fam_stud
Adding user `fam_stud' ...
Adding new group `fam_stud' (1001) ...
Adding new user `fam_stud' (1001) with group `fam_stud' ...
Creating home directory `/home/fam_stud' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for fam_stud
Enter the new value, or press ENTER for the default
    Full Name []: Student
    Room Number []: 00
    Work Phone []: 00
    Home Phone []: 999
    Other []: 111
Is the information correct? [Y/n]
root@deb10-stud:/sbin#
```

Добавить пользователя в группу sudo.

После перезапуска системы строка приглашения и окно терминала должны иметь вид:



```
fam_stud@ab-n: ~
File Edit View Search Terminal Help
fam_stud@ab-n: ~$
```

Определить список источников для менеджера пакетов APT (Advanced Packaging Tool).

```

/etc/apt/sources.list [----] 0 L:[ 1+ 9 10/ 28] *(532 /1221b) 0100 0x064 [*][X]
#
# deb cdrom:[Debian GNU/Linux 10.7.0 _Buster_ - Official i386 NETINST 20201205-12:46]/ buster main
#deb cdrom:[Debian GNU/Linux 10.7.0 _Buster_ - Official i386 NETINST 20201205-12:46]/ buster main
deb cdrom:[Debian GNU/Linux 10.8.0 _Buster_ - Official i386 DVD Binary-3 20210206-10:47]/ buster contrib main
deb cdrom:[Debian GNU/Linux 10.8.0 _Buster_ - Official i386 DVD Binary-2 20210206-10:47]/ buster contrib main
deb cdrom:[Debian GNU/Linux 10.8.0 _Buster_ - Official i386 DVD Binary-1 20210206-10:47]/ buster contrib main
deb http://deb.debian.org/debian/ buster main
# deb-src http://deb.debian.org/debian/ buster main

# deb http://security.debian.org/debian-security buster/updates main
# deb-src http://security.debian.org/debian-security buster/updates main

# buster-updates, previously known as 'volatile'
# deb http://deb.debian.org/debian/ buster-updates main
# deb-src http://deb.debian.org/debian/ buster-updates main

# This system was installed using small removable media
# (e.g. netinst, live or single CD). The matching "deb cdrom"
# entries were disabled at the end of the installation process.
# For information about how to configure apt package sources,
# see the sources.list(5) manual.

1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn 10Quit

```

Диски DVD binary добавлены с помощью утилиты **apt-cdrom** . Проверить и установить обновления - **update** , **upgrade** .
Проверить установку компилятора GCC

```

fam_stud@ab-n: ~
File Edit View Search Terminal Help
fam_stud@ab-n:~$ sudo apt-get install gcc
Reading package lists... Done
Building dependency tree
Reading state information... Done
gcc is already the newest version (4:8.3.0-1).
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
fam_stud@ab-n:~$

```

Допускается использовать команду: *apt-get install gcc-multilib*.

Установить отладчик :

```

fam_stud@ab-n: ~
File Edit View Search Terminal Help
fam_stud@ab-n:~$ sudo apt-get install gdb
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libbabeltrace1 libc6-dbg libipt2
Suggested packages:
  gdb-doc gdbserver
The following NEW packages will be installed:
  gdb libbabeltrace1 libc6-dbg libipt2
0 upgraded, 4 newly installed, 0 to remove and 37 not upgraded.
Need to get 0 B/22.0 MB of archives.
After this operation, 36.7 MB of additional disk space will be used.
Do you want to continue? [Y/n]

```

Установить графический интерфейс отладчика KDbg

```

root@ab-n:/etc/apt# sudo apt install kdbg
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
-----
0 upgraded, 199 newly installed, 0 to remove and 37 not upgraded.
Need to get 327 kB/127 MB of archives.
After this operation, 426 MB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 http://deb.debian.org/debian buster/main i386 kdbg i386 2.5.5-3 [327 kB]
Media change: please insert the disc labeled
'Debian GNU/Linux 10.8.0 _Buster_ - Official i386 DVD Binary-1 20210206-10:47'
in the drive '/media/cdrom/' and press [Enter]
-----
Setting up kdbg (2.5.5-3) ...
Processing triggers for libvlc-bin:i386 (3.0.12-0+deb10u1) ...
Processing triggers for libc-bin (2.28-10) ...
root@ab-n:/etc/apt# █

```

По завершению установки система готова и можно приступать к выполнению заданий.

2. Переменные окружения

Окружение представляет собой набор пар вида 'имя-значение' для каждой программы. Эти пары называются переменными окружения. Каждое имя состоит от одной до любого числа буквенно-цифровых символов или символов подчеркивания ('_'), но имя не может начинаться с цифры. (Это правило контролируется оболочкой; С API может помещать в окружение все, что захочет, за счет возможного запутывания последующих программ.)

Переменные окружения часто используются для управления поведением программ.

Преимуществом использования переменных окружения является то, что пользователи могут установить их в своем загрузочном файле и не беспокоиться больше постоянным набором определенных опций в командной строке.

Внешняя переменная `environ` является официальным, стандартным, переносимым способом получения доступа ко всему окружению предоставляет доступ таким же способом, как `argv` (argument vector) предоставляет доступ к аргументам командной строки, `argc` (argument count) - это количество аргументов командной строки.

Пример программы.

```

/* printenv.c --- Print out the environment. */
#include <stdio.h>

extern char **environ;

int main(int argc, char **argv)
{
    int i;

    if (environ != NULL)
        for (i = 0; environ[i] != NULL; i++)
            printf("%s\n", environ[i]);

    return 0;
}

```

Выполнить компиляцию. Изучить и дать пояснения значениям параметров и флагов.

```
fam_stud@ab-n: ~/Test
File Edit View Search Terminal Help
fam_stud@ab-n:~/Test$ ls
printenv.c
fam_stud@ab-n:~/Test$ sudo gcc -g -Wall -Wextra printenv.c -o printenv
[sudo] password for fam_stud:
printenv.c: In function 'main':
printenv.c:7:14: warning: unused parameter 'argc' [-Wunused-parameter]
  int main(int argc, char **argv)
             ~~~~~^
printenv.c:7:27: warning: unused parameter 'argv' [-Wunused-parameter]
  int main(int argc, char **argv)
                   ~~~~~^
fam_stud@ab-n:~/Test$ ls
printenv  printenv.c
fam_stud@ab-n:~/Test$
```

Результат выполнения. Провести анализ и дать пояснения значениям параметров (несколько 3-5).

```
Sun 12:31
fam_stud@ab-n: ~/Test
File Edit View Search Terminal Help
fam_stud@ab-n:~/Test$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/ab-n:0/tmp/.ICE-unix/1274,unix/ab-n:0/tmp/.ICE-unix/1274
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LANGUAGE=en_US:en
SSH_AUTH_SOCK=/run/user/1001/keyring/ssh
DESKTOP_SESSION=gnome
GTK_MODULES=gail:atk-bridge
XDG_SEAT=seat0
PWD=/home/fam_stud/Test
XDG_SESSION_DESKTOP=gnome
LOGNAME=fam_stud
XDG_SESSION_TYPE=wayland
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
GDM_LANG=en_US.UTF-8
HOME=/home/fam_stud
USERNAME=fam_stud
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34;ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33:cd=40;33:or=40;31:01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzt=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcc=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
:
XDG_CURRENT_DESKTOP=GNOME
VTE_VERSION=5402
WAYLAND_DISPLAY=wayland-0
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/7bb131e0_a209_419c_abe3_4dfefbae266f
GJS_DEBUG_OUTPUT=stderr
XDG_SESSION_CLASS=user
TERM=xterm-256color
USER=fam_stud
GNOME_TERMINAL_SERVICE=:1.56
DISPLAY=:0
SHLVL=1
XDG_VTNR=2
XDG_SESSION_ID=2
XDG_RUNTIME_DIR=/run/user/1001
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
GDMSESSION=gnome
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1001/bus
_/usr/bin/printenv
OLDPWD=/home/fam_stud
fam_stud@ab-n:~/Test$
```

Результат выполнения программы можно перенаправить в файл.

```
fam_stud@ab-n: ~/Test
File Edit View Search Terminal Help
fam_stud@ab-n:~/Test$ printenv > printenv.txt
fam_stud@ab-n:~/Test$ ls
printenv  printenv.c  printenv.txt
fam_stud@ab-n:~/Test$
```

```
Sun 13:31
printenv.txt
~/Test
Save
SHELL=/bin/bash
SESSION_MANAGER=local/ab-n:0/tmp/.ICE-unix/1274,unix/ab-n:0/tmp/.ICE-unix/1274
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LANGUAGE=en_US:en
SSH_AUTH_SOCK=/run/user/1001/keyring/ssh
DESKTOP_SESSION=gnome
GTK_MODULES=gail:atk-bridge
XDG_SEAT=seat0
PWD=/home/fam_stud/Test
XDG_SESSION_DESKTOP=gnome
LOGNAME=fam_stud
XDG_SESSION_TYPE=wayland
Plain Text Tab Width: 8 Ln1, Col1 INS
```

Последовательность выполнения кода программы следует изучить с использованием консольного отладчика.

```
Sun 13:41
fam_stud@ab-n: ~/Test
File Edit View Search Terminal Help
fam_stud@ab-n:~/Test$ gdb printenv
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 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 "i686-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/>.

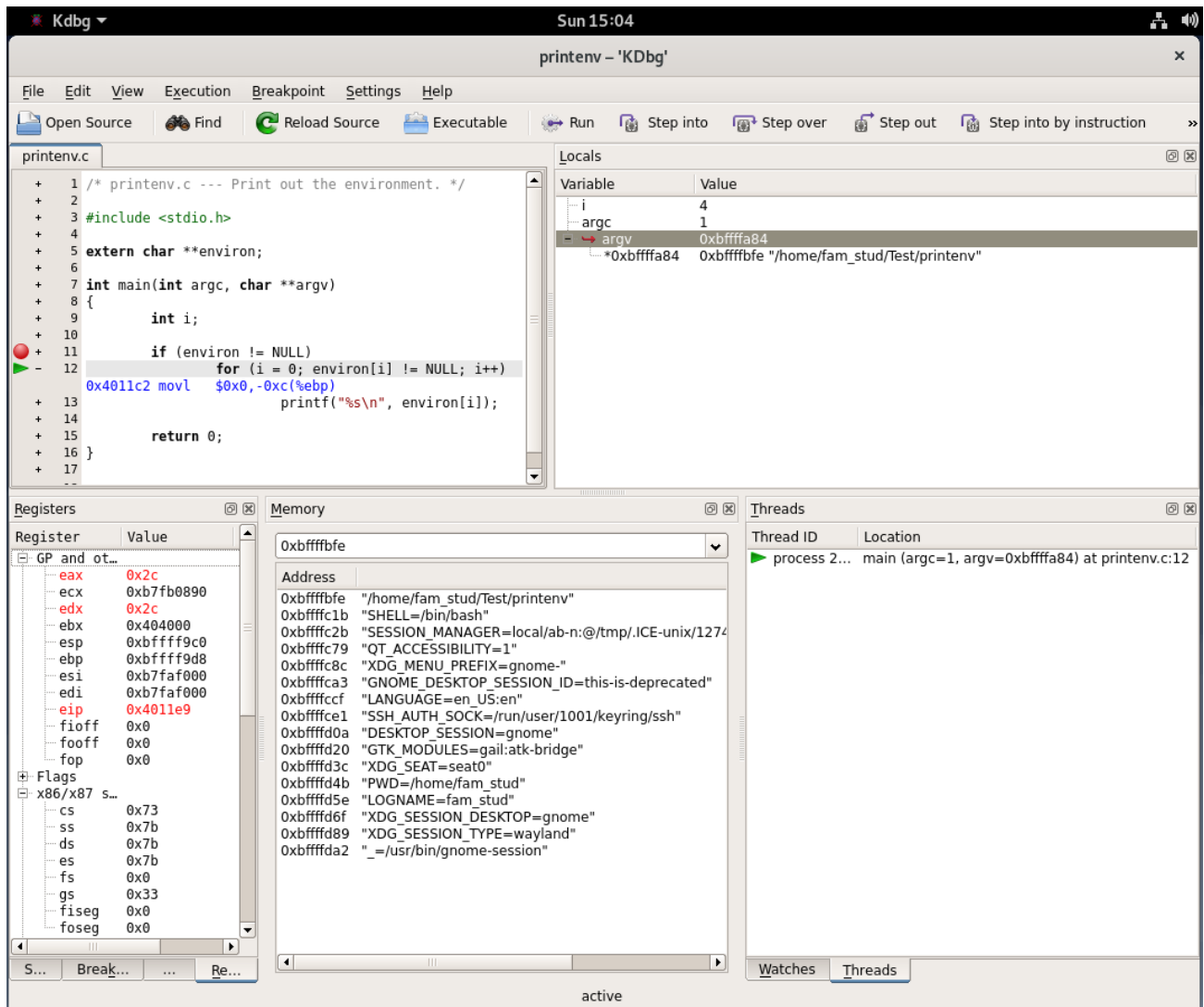
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from printenv...done.
(gdb)
(gdb) list
1      /* printenv.c --- Print out the environment. */
2
3      #include <stdio.h>
4
5      extern char **environ;
6
7      int main(int argc, char **argv)
8      {
9          int i;
10
11         if (environ != NULL)
12             for (i = 0; environ[i] != NULL; i++)
13                 printf("%s\n", environ[i]);
14
15         return 0;
16     }
17
(gdb) list
Line number 18 out of range; printenv.c has 17 lines.
(gdb)
```

Выполнить дизассемблирование и проанализировать код.

```
(gdb) disassemble main
Dump of assembler code for function main:
0x00001199 <+0>:    lea    0x4(%esp),%ecx
0x0000119d <+4>:    and    $0xffffffff,%esp
0x000011a0 <+7>:    pushl  -0x4(%ecx)
0x000011a3 <+10>:   push  %ebp
0x000011a4 <+11>:   mov    %esp,%ebp
0x000011a6 <+13>:   push  %ebx
0x000011a7 <+14>:   push  %ecx
0x000011a8 <+15>:   sub    $0x10,%esp
0x000011ab <+18>:   call   0x10a0 <_x86.get_pc_thunk.bx>
0x000011b0 <+23>:   add    $0x2e50,%ebx
0x000011b6 <+29>:   mov    -0x4(%ebx),%eax
0x000011bc <+35>:   mov    (%eax),%eax
0x000011be <+37>:   test   %eax,%eax
0x000011c0 <+39>:   je     0x1203 <main+106>
0x000011c2 <+41>:   movl   $0x0,-0xc(%ebp)
0x000011c9 <+48>:   jmp    0x11ed <main+84>
0x000011cb <+50>:   mov    -0x4(%ebx),%eax
0x000011d1 <+56>:   mov    (%eax),%eax
0x000011d3 <+58>:   mov    -0xc(%ebp),%edx
0x000011d6 <+61>:   shl    $0x2,%edx
--Type <RET> for more, q to quit, c to continue without paging--
0x000011d9 <+64>:   add    %edx,%eax
0x000011db <+66>:   mov    (%eax),%eax
0x000011dd <+68>:   sub    $0xc,%esp
0x000011e0 <+71>:   push  %eax
0x000011e1 <+72>:   call   0x1030 <puts@plt>
0x000011e6 <+77>:   add    $0x10,%esp
0x000011e9 <+80>:   addl   $0x1,-0xc(%ebp)
0x000011ed <+84>:   mov    -0x4(%ebx),%eax
0x000011f3 <+90>:   mov    (%eax),%eax
0x000011f5 <+92>:   mov    -0xc(%ebp),%edx
0x000011f8 <+95>:   shl    $0x2,%edx
0x000011fb <+98>:   add    %edx,%eax
0x000011fd <+100>:  mov    (%eax),%eax
0x000011ff <+102>:  test   %eax,%eax
0x00001201 <+104>:  jne    0x11cb <main+50>
0x00001203 <+106>:  mov    $0x0,%eax
0x00001208 <+111>:  lea    -0x8(%ebp),%esp
0x0000120b <+114>:  pop    %ecx
0x0000120c <+115>:  pop    %ebx
0x0000120d <+116>:  pop    %ebp
0x0000120e <+117>:  lea    -0x4(%ecx),%esp
--Type <RET> for more, q to quit, c to continue without paging--
0x00001211 <+120>:  ret
End of assembler dump.
(gdb)
```

В ряде случаев удобно использовать приложение KDbg и используя точки останова выполнения программы проанализировать состояние регистров процессора разделов памяти.

После загрузки программы установить точки останова и запустить выполнение.



Пошаговое выполнение кода позволяет проанализировать исполнение команд, сравнивая изменения значения переменных, содержимого регистров процессора и разделов памяти. Требуется выполнить 2-3 шага.

3. Исследование адресного пространства Linux/Unix

Программа, `memaddr.c` распечатывает местонахождение двух функций `main()` и `afunc()` (строки 22-23). Затем она показывает, как стек растет вниз, позволяя `afunc()` (строки 51-63) распечатать адреса последовательных экземпляров ее локальной переменной `stack_var`. (`stack_var` намеренно объявлена как `auto`, чтобы подчеркнуть, что она находится в стеке.) Затем она показывает расположение памяти, выделенной с помощью `alloca()` (строки 28-32). В заключение она печатает местоположение переменных данных и BSS (строки 34-38), а затем памяти, выделенной непосредственно через `sbrk()` (строки 40-48).

```
/*
 * memaddr.c --- Show address of code, data and stack sections,
 *               as well as BSS and dynamic memory.
 */

#include <stdio.h>
#include <malloc.h>          /* for definition of ptrdiff_t on GLIBC */
#include <unistd.h>
```



```

#include <alloca.h>          /* for demonstration only */

extern void afunc(void);     /* a function for showing stack growth */

int bss_var;                /* auto init to 0, should be in BSS */
int data_var = 42;          /* init to nonzero, should be data */

int
main(int argc, char **argv) /* arguments aren't used */
{
    char *p, *b, *nb;

    printf("Text Locations:\n");
    printf("\tAddress of main: %p\n", main);
    printf("\tAddress of afunc: %p\n", afunc);

    printf("Stack Locations:\n");
    afunc();

    p = (char *) alloca(32);
    if (p != NULL) {
        printf("\tStart of alloca()'ed array: %p\n", p);
        printf("\tEnd of alloca()'ed array: %p\n", p + 31);
    }

    printf("Data Locations:\n");
    printf("\tAddress of data_var: %p\n", &data_var);

    printf("BSS Locations:\n");
    printf("\tAddress of bss_var: %p\n", &bss_var);

    b = sbrk((ptrdiff_t) 32);    /* grow address space */
    nb = sbrk((ptrdiff_t) 0);
    printf("Heap Locations:\n");
    printf("\tInitial end of heap: %p\n", b);
    printf("\tNew end of heap: %p\n", nb);

    b = sbrk((ptrdiff_t) -16);   /* shrink it */
    nb = sbrk((ptrdiff_t) 0);
    printf("\tFinal end of heap: %p\n", nb);
}

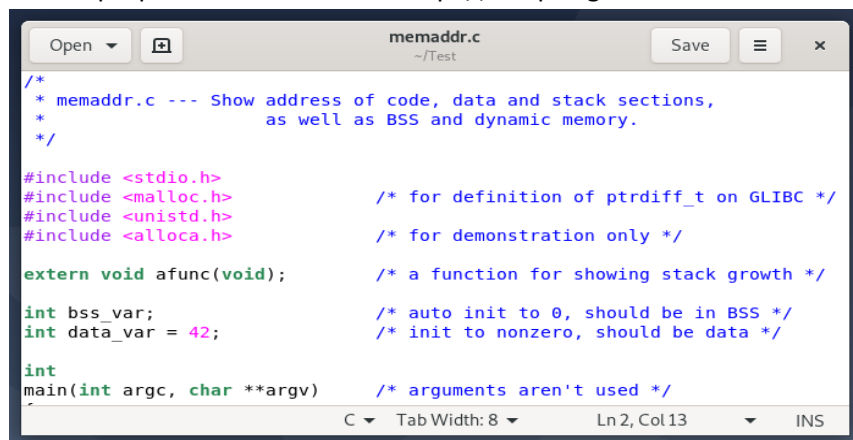
void
afunc(void)
{
    static int level = 0;        /* recursion level */
    auto int stack_var;         /* automatic variable, on stack */

    if (++level == 3)           /* avoid infinite recursion */
        return;

    printf("\tStack level %d: address of stack_var: %p\n",
           level, &stack_var);
    afunc();                    /* recursive call */
}

```

Текст программы можно ввести в редакторах gedit



```

/*
 * memaddr.c --- Show address of code, data and stack sections,
 *               as well as BSS and dynamic memory.
 */

#include <stdio.h>
#include <malloc.h>          /* for definition of ptrdiff_t on GLIBC */
#include <unistd.h>
#include <alloca.h>          /* for demonstration only */

extern void afunc(void);     /* a function for showing stack growth */

int bss_var;                /* auto init to 0, should be in BSS */
int data_var = 42;          /* init to nonzero, should be data */

int
main(int argc, char **argv) /* arguments aren't used */
{
    char *p, *b, *nb;

    printf("Text Locations:\n");
    printf("\tAddress of main: %p\n", main);
    printf("\tAddress of afunc: %p\n", afunc);

    printf("Stack Locations:\n");
    afunc();

    p = (char *) alloca(32);
    if (p != NULL) {
        printf("\tStart of alloca()'ed array: %p\n", p);
        printf("\tEnd of alloca()'ed array: %p\n", p + 31);
    }

    printf("Data Locations:\n");
    printf("\tAddress of data_var: %p\n", &data_var);

    printf("BSS Locations:\n");
    printf("\tAddress of bss_var: %p\n", &bss_var);

    b = sbrk((ptrdiff_t) 32);    /* grow address space */
    nb = sbrk((ptrdiff_t) 0);
    printf("Heap Locations:\n");
    printf("\tInitial end of heap: %p\n", b);
    printf("\tNew end of heap: %p\n", nb);

    b = sbrk((ptrdiff_t) -16);   /* shrink it */
    nb = sbrk((ptrdiff_t) 0);
    printf("\tFinal end of heap: %p\n", nb);
}

void
afunc(void)
{
    static int level = 0;        /* recursion level */
    auto int stack_var;         /* automatic variable, on stack */

    if (++level == 3)           /* avoid infinite recursion */
        return;

    printf("\tStack level %d: address of stack_var: %p\n",
           level, &stack_var);
    afunc();                    /* recursive call */
}

```

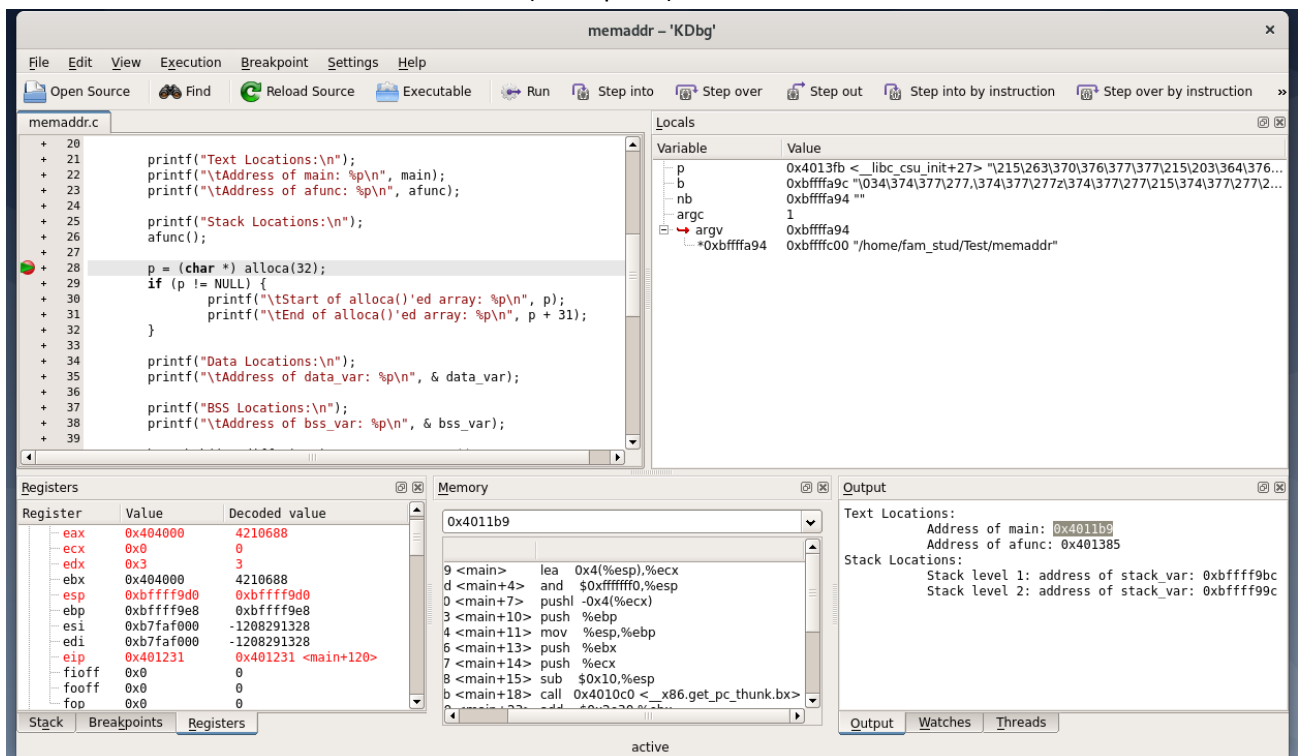
Результат запуска программы на системе Intel GNU/Linux:

```
fam_stud@ab-n: ~/Test
File Edit View Search Terminal Help
fam_stud@ab-n:~/Test$
fam_stud@ab-n:~/Test$ sudo gcc -g memaddr.c -o memaddr
[sudo] password for fam_stud:
fam_stud@ab-n:~/Test$ ls
memaddr memaddr.c printenv printenv.c printenv.txt
fam_stud@ab-n:~/Test$ ./memaddr
Text Locations:
  Address of main: 0x4721b9
  Address of afunc: 0x472385
Stack Locations:
  Stack level 1: address of stack_var: 0xbfe238ac
  Stack level 2: address of stack_var: 0xbfe2388c
  Start of alloca()'ed array: 0xbfe23890
  End of alloca()'ed array: 0xbfe238af
Data Locations:
  Address of data_var: 0x475024
BSS Locations:
  Address of bss_var: 0x475030
Heap Locations:
  Initial end of heap: 0x8e7000
  New end of heap: 0x8e7020
  Final end of heap: 0x8e7010
fam_stud@ab-n:~/Test$
```

Далее следует исследовать выполнение программы.

Загрузить исходный код программы(Source) и исполняемый файл (Executable) в отладчик KDbg.

Установить точки останова исполнения (breakpoint).



Выполнить пошаговое выполнение программы и с учетом описания работы программы сравнить состояния регистров с результатами полученными при запуске программы в командной строке либо в окне KDbg Program output.

При анализе выполнения следует учитывать, что при разработке эксплойтов (как и других программ) используются в основном регистры общего назначения, поэтому важно понимать их назначение. Этими регистрами являются:

`eax` (Extended Account — расширенный сумматор);

`ebx` (Extended Base расширенный регистр базы);

`ecx` (Extended Account — расширенный сумматор);

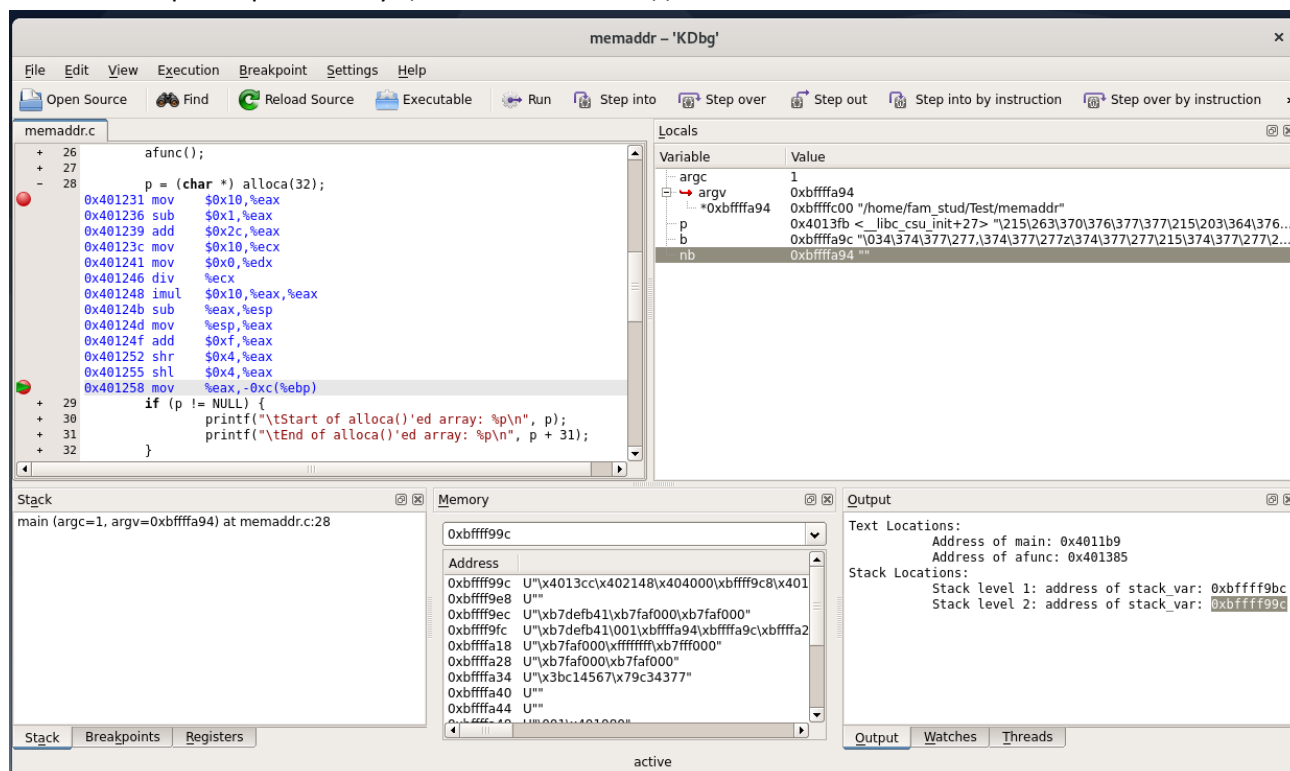
`ebx` (Extended Base расширенный регистр базы);

ecx (Extended Counter — расширенный счетчик);
edx (Extended Data — расширенный регистр данных);
esp (Extended Stack Pointer — расширенный указатель стека);
ebp (Extended Base Pointer — расширенный указатель базы);
esi (Extended Source Index — расширенный индекс источника);
edi (Extended Destination Index — расширенный индекс назначения).

Эти регистры оперируют 32-битовыми значениями.

Регистр флагов E FLAGS/FLAGS и регистр указатель команды EIP/IP) содержат информацию о состоянии процессора исполняемой программы и позволяют изменить это состояние.

EIP/IP (Instruction Pointer register) — регистр-указатель команд. Регистр eip/ip имеет разрядность 32/16 бит и содержит смещение следующей подлежащей выполнению команды относительно содержимого сегментного регистра cs в текущем сегменте команд.



В данном случае, как происходит выполнение строки кода `p = (char *) alloca(32);`. Функция **alloca** выделяет 32 байт памяти в стеке.

4. Использование уязвимостей: стек

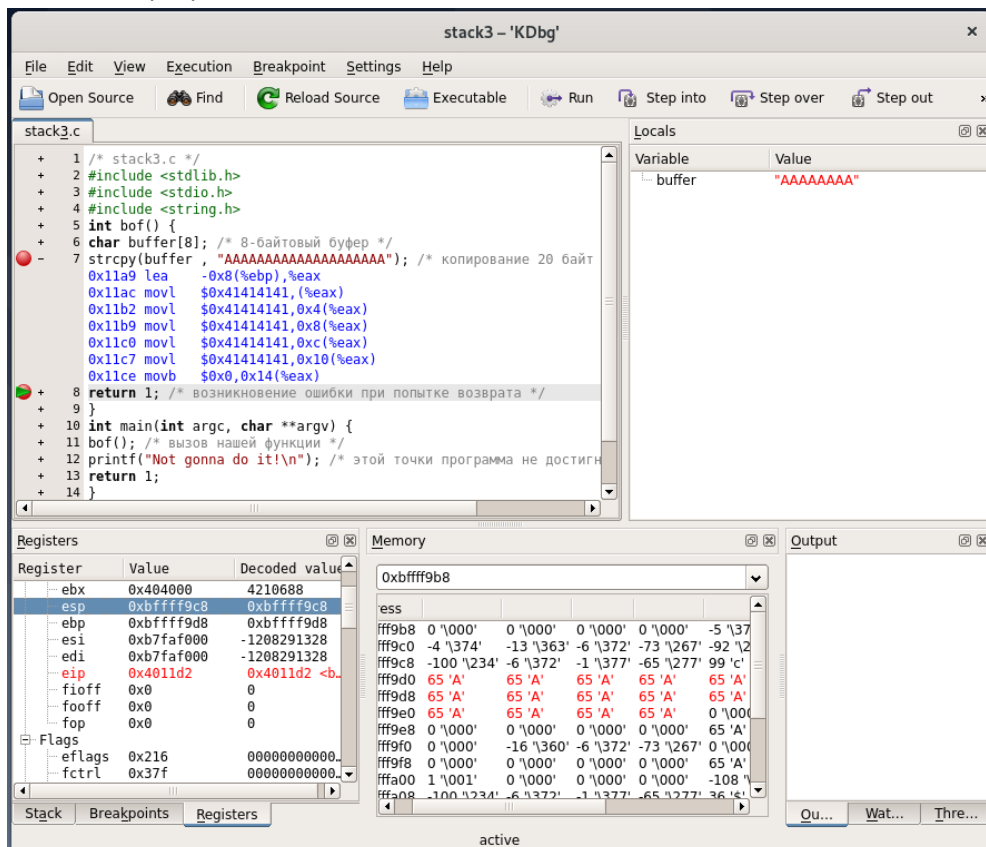
Причиной появления уязвимостей переполнения буфера является то, что данные программ не отделены от структур, управляющих как данными, так и потоком выполнения программы. Если в выделенный программой буфер копируются данные, длина которых превышает его размер, лишние байты переписывают другие данные в прилегающей к буферу области. Переполнение стека возникает тогда, когда такой буфер расположен в стеке. При этом может быть переписан сохраненный ранее адрес возврата из функции. Если копируемые данные подготовить специальным образом, можно

переместить выполнение программы на код, помещенный атакующим в буфер. Например, в UNIX можно заставить программу с привилегиями суперпользователя выполнить системный вызов, запускающий командную оболочку с такими же привилегиями. Атаки этого типа могут быть проведены локально, путем предоставления поддельных исходных данных программе с интерактивным вводом.

Код, приведенный ниже, является примером программы с неконтролируемым переполнением, где продемонстрированы распространенная программная ошибка и негативные последствия для безопасности программы, к которым такого рода ошибка может привести:

```
/* stack3.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof() {
    char buffer[8]; /* 8-байтовый буфер */
    strcpy(buffer, "AAAAAAAAAAAAAAAAAAAA"); /* копирование 20 байт в буфер при выполнении
    вместо "А" ввести Фамилия_Группа_Номер по списку Студента */
    return 1; /* возникновение ошибки при попытке возврата */
}
int main(int argc, char **argv) {
    bof(); /* вызов нашей функции */
    printf("Not gonna do it!\n"); /* этой точки программа не достигнет */
    return 1;
}
```

В программе вызывается пользовательская функция bof. В этой функции строка из 20 букв "А" копируется в буфер, рассчитанный на хранение 8 байт, что приводит к его переполнению. Заметьте, что до вызова функции print/ программа не доходит переполнение буфера приведет к тому, что при попытке возврата из функции bof управление не будет передано обратно в функцию main. Выполнение программы в отладчике.



5. Программа с переполнением буфера

Сначала необходимо создать уязвимую программу и понять, почему она уязвима и каким образом этой уязвимостью можно воспользоваться. Рассматриваемая программа подобна приведенной в предыдущем примере, с той лишь разницей, что данные для переполнения могут быть введены пользователем, а не предопределены символьной строкой в программе. Таким образом мы сможем контролировать изменение адреса возврата и дальнейшие действия.

Программа будет считывать входные данные из файла в небольшой буфер, размещенный в стеке. Если объем данных превысит размер буфера, случится переполнение стека. А поскольку содержимое файла можно изменять, это предоставит нам возможность изучить, каким образом

переполнение стека можно использовать. Для создания новой программы достаточно заменить функцию bof предыдущего примера:

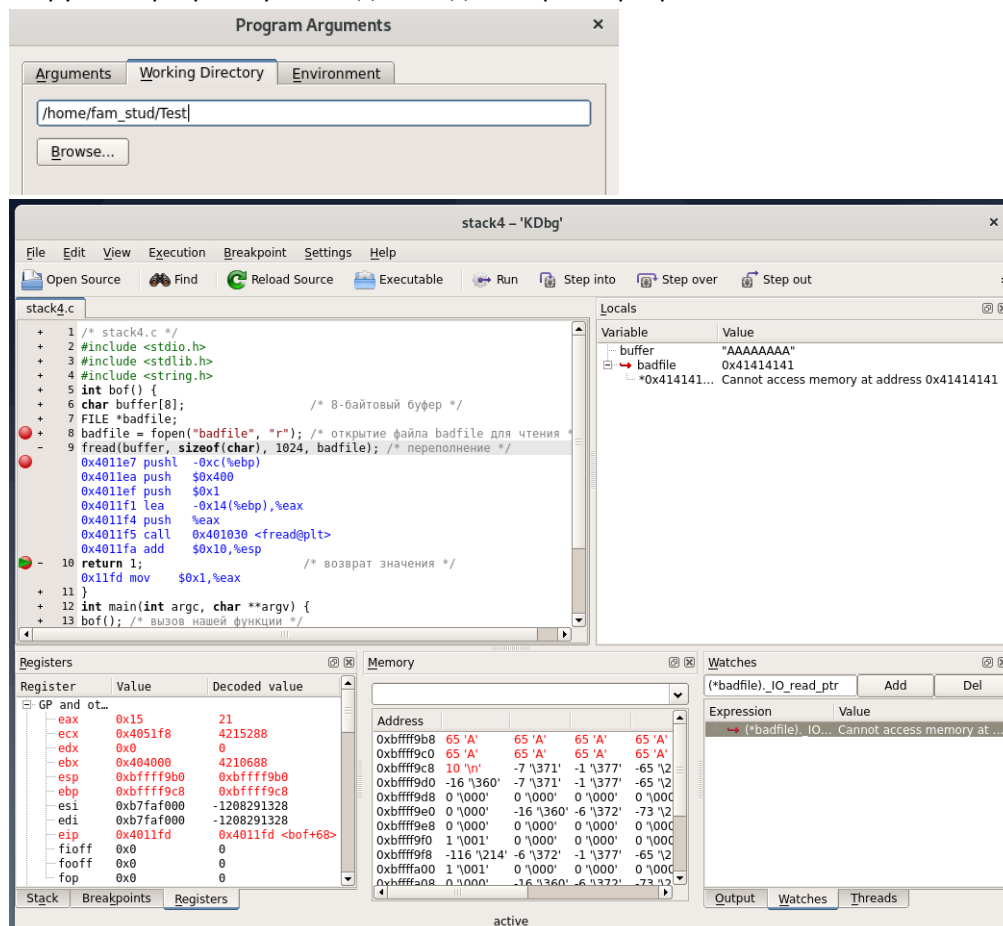
```
/* stack4.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int bof() {
    char buffer[8];          /* 8-байтовый буфер */
    FILE *badfile;
    badfile = fopen("badfile", "r"); /* открытие файла badfile для чтения */
    fread(buffer, sizeof(char), 1024, badfile); /* переполнение */
    return 1;                /* возврат значения */
}
int main(int argc, char **argv) {
    bof(); /* вызов нашей функции */
    printf("Not gonna do it!\n"); /* этой точки программа не достигнет */
    return 1;
}
```



При выполнении вместо "А" ввести *Фамилия Группа Номер по списку Студента.*

В этой функции открывается файл с именем badfile (имя файла predetermined для упрощения примера), далее делается попытка считать из него 1024 байта — в 8-байтовый буфер, затем файл закрывается. При выходе из функции должно произойти переполнение буфера. Таким образом, содержимое считываемого файла даст возможность контролировать адрес возврата в стековом кадре функции bof.

Загрузить программу в отладчик задать параметры рабочего каталога .



Для сравнения можно изменять количество букв в файле и сравнить результат выполнения программы. Состояние стека после переполнения похоже на то, что было в предыдущей программе. Разница только в том, что в стеке выделено место для еще одной переменной — файлового дескриптора. Представить результат сравнения.

6. Использование уязвимостей: куча

Адресное пространство для кучи выделяется, как правило, в том же сегменте, что и для стека, и заполняется в направлении стека от старших адресов к младшим. Память и куче выделяется с помощью функций типа `malloc`, существующих во всех языках структурного программирования — `HeapAlloc` в Windows, `malloc` в стандарте ANSI для языка C или `new` в C++. Соответствующие функции `HeapFree`, `free` и `delete` существуют, естественно, и для освобождения памяти. Всю реальную работу по распределении памяти выполняет компонент операционной системы или стандартной библиотеки языка C, известный как диспетчер кучи. Именно он отвечает за выделение памяти для процессов и управляет ростом кучи. Таким образом, если процессу требуется больше динамической памяти, он ее получает.

6.1. Простое переполнение кучи

Если говорить упрощенно, куча состоит из множества участков памяти (рис. 1).

Некоторые из них выделены для использования программой, другие свободны.

Выделенные участки часто размещаются в памяти рядом.

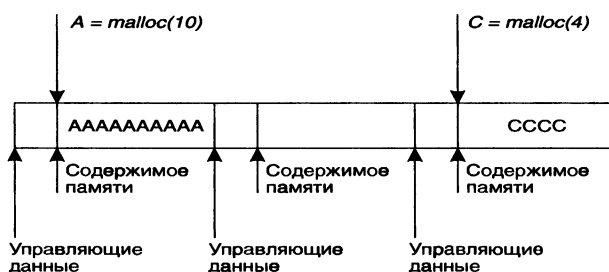


Рис. 1. Упрощенное представление кучи

Далее приведен пример уязвимой программы, использующей память в куче. Программа содержит ошибку, позволяющую организовать переполнение буфера:

```
/* heap1.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char *input = malloc(20);
    char *output = malloc(20);

    strcpy(output, "normal output");
    strcpy(input, argv[1]);
    printf("input at %p %s\n", input, input);
    printf("output at %p %s\n", output, output);
    printf("\n%s\n", output);
}
```

Рассмотрим, что происходит, когда в буфер `input` помещается больше данных, чем изначально было выделено пространства. Это становится возможным, если контроль над размером заносящихся в буфер данных не производится (строка 11).

Таким образом, достичь переполнения в куче очень просто, причем это не всегда вызывает сбой в программе. На рис. 2 показано, как все происходит.

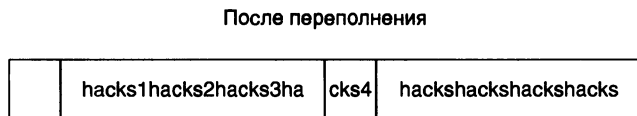


Рис. 2. Механизм переполнения кучи

Пример компиляции и выполнения программы:

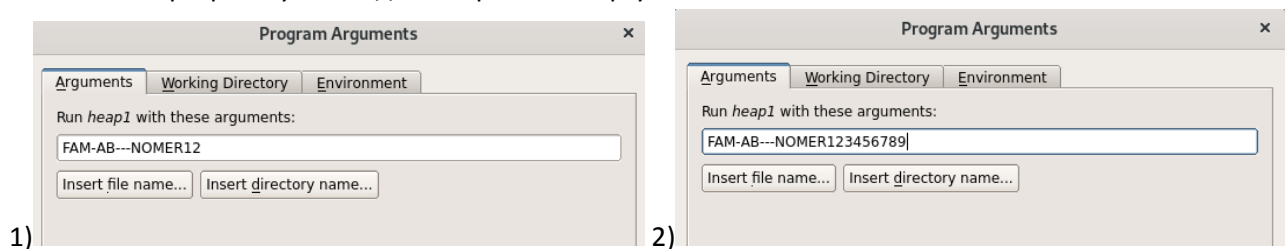
fam_stud@ab-n:~/Test\$ sudo gcc -g heap1.c -o heap1

```
fam_stud@ab-n: ~/Test
File Edit View Search Terminal Help
fam_stud@ab-n:~/Test$ ./heap1 FAM-AB---NOMER
input at 0xe72160 FAM-AB---NOMER
output at 0xe72180 normal output

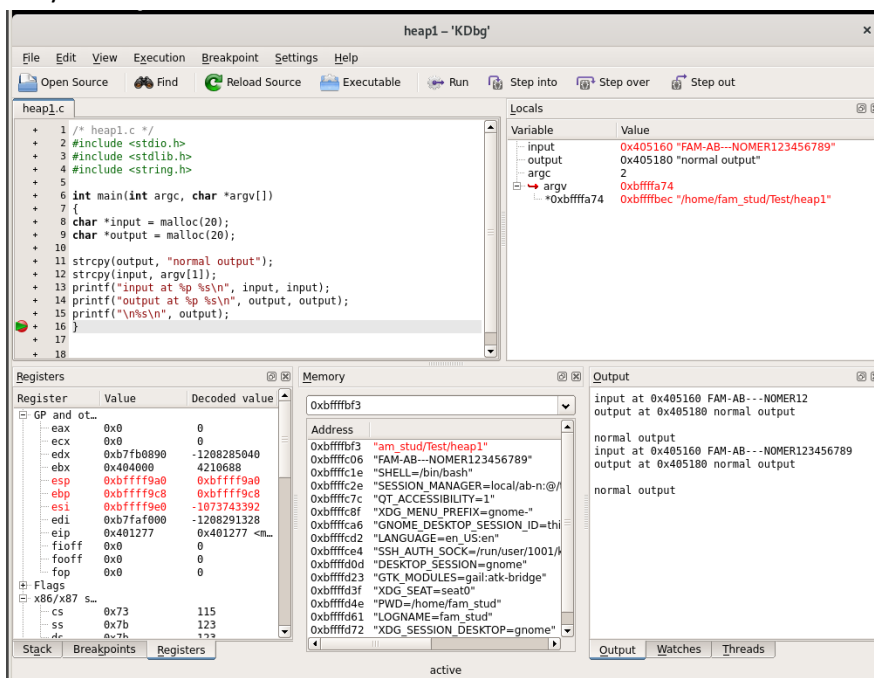
normal output
fam_stud@ab-n:~/Test$ ./heap1 FAM-AB---NOMER123456789
input at 0x5c6160 FAM-AB---NOMER123456789
output at 0x5c6180 normal output

normal output
fam_stud@ab-n:~/Test$
```

Выполнить программу в отладчике с разными аргументами.



Результат выполнения



Отладчик показывает переполнения в куче, причем это не вызывает сбой в программе.

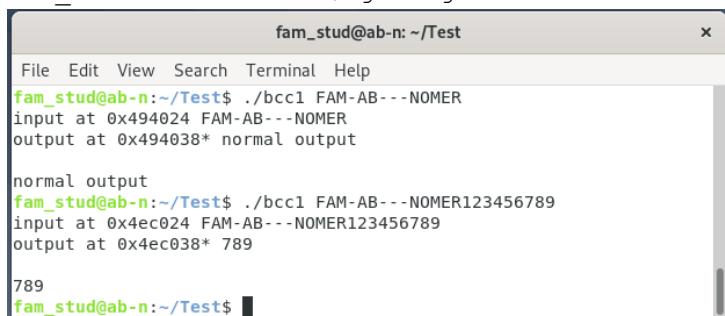
6.2. Переполнение в сегменте Бсс

Подобное переполнение может быть выполнено и для статических переменных, размещенных в сегменте Бсс. Рассмотрим, как это может выглядеть в примере, приближенном к реальности:

```
/* bcc1.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
static char input[20];
static char output[20];
int main(int argc, char *argv[]) {
    strcpy(output, "normal output");
    strcpy(input, argv[1]);
    printf ("input at %p %s\n" , input, input);
    printf ("output at %p %s\n", output, output);
    printf("\n%s\n", output);
}
```

Пример компиляции выполнения программы:

```
fam_stud@ab-n:~/Test$ gcc -g bcc1.c -o bcc1
```



Использование консольного отладчика:

```
am_stud@ab-n:~/Test$ gdb bcc1
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 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 "i686-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 bcc1...done.
(gdb) list
1      /* bcc1.c */
2      #include <stdio.h>
3      #include <stdlib.h>
4      #include <string.h>
5      static char input[20];
6      static char output[20];
7      int main(int argc, char *argv[]) {
8          strcpy(output, "normal output");
9          strcpy(input, argv[1]);
10         printf ("input at %p %s\n" , input, input);
(gdb) break 10
Breakpoint 1 at 0x1207: file bcc1.c, line 10.
(gdb) run FAM-AB---NOMER123456789
Starting program: /home/fam_stud/Test/bcc1 FAM-AB---NOMER123456789

Breakpoint 1, main (argc=2, argv=0xbffff424) at bcc1.c:10
10         printf ("input at %p %s\n" , input, input);
(gdb) info frame
Stack level 0, frame at 0xbffff390:
eip = 0x401207 in main (bcc1.c:10); saved eip = 0xb7defb41
```



```

source language c.
Arglist at 0xbffff378, args: argc=2, argv=0xbffff424
Locals at 0xbffff378, Previous frame's sp is 0xbffff390
Saved registers:
  ebx at 0xbffff374, ebp at 0xbffff378, eip at 0xbffff38c
(gdb) info proc mappings
process 2095
Mapped address spaces:

      Start Addr   End Addr       Size     Offset objfile
      0x400000     0x401000     0x1000        0x0  /home/fam_stud/Test/bcc1
      0x401000     0x402000     0x1000       0x1000  /home/fam_stud/Test/bcc1
      0x402000     0x403000     0x1000       0x2000  /home/fam_stud/Test/bcc1
      0x403000     0x404000     0x1000       0x2000  /home/fam_stud/Test/bcc1
      0x404000     0x405000     0x1000       0x3000  /home/fam_stud/Test/bcc1
      0xb7dd5000   0xb7dee000     0x19000        0x0  /usr/lib/i386-linux-gnu/libc-2.28.so
      0xb7dee000   0xb7f3c000    0x14e000     0x19000  /usr/lib/i386-linux-gnu/libc-2.28.so
      0xb7f3c000   0xb7fac000    0x70000     0x167000  /usr/lib/i386-linux-gnu/libc-2.28.so
      0xb7fac000   0xb7fad000     0x1000     0x1d7000  /usr/lib/i386-linux-gnu/libc-2.28.so
      0xb7fad000   0xb7faf000     0x2000     0x1d7000  /usr/lib/i386-linux-gnu/libc-2.28.so
      0xb7faf000   0xb7fb0000     0x1000     0x1d9000  /usr/lib/i386-linux-gnu/libc-2.28.so
      0xb7fb0000   0xb7fb3000     0x3000        0x0
      0xb7fcf000   0xb7fd1000     0x2000        0x0
      0xb7fd1000   0xb7fd4000     0x3000        0x0 [vvar]
      0xb7fd4000   0xb7fd6000     0x2000        0x0 [vdso]
      0xb7fd6000   0xb7fd7000     0x1000        0x0  /usr/lib/i386-linux-gnu/ld-2.28.so
      0xb7fd7000   0xb7ff3000    0x1c000     0x1000  /usr/lib/i386-linux-gnu/ld-2.28.so
      0xb7ff3000   0xb7ffd000     0xa000     0x1d000  /usr/lib/i386-linux-gnu/ld-2.28.so
      0xb7ffe000   0xb7fff000     0x1000     0x27000  /usr/lib/i386-linux-gnu/ld-2.28.so
      0xb7fff000   0xb8000000     0x1000     0x28000  /usr/lib/i386-linux-gnu/ld-2.28.so
      0xbffdf000   0xc0000000    0x21000        0x0 [stack]
(gdb) info all-registers
eax             0x404024             4210724
ecx             0xbffff5d0          -1073744432
edx             0x40402e             4210734
ebx             0x404000             4210688
esp             0xbffff370          0xbffff370
ebp             0xbffff378          0xbffff378
esi             0xb7faf000          -1208291328
edi             0xb7faf000          -1208291328
eip             0x401207             0x401207 <main+94>
eflags          0x282             [ SF IF ]
cs              0x73              115
ss              0x7b              123
ds              0x7b              123
es              0x7b              123
fs              0x0              0
gs              0x33              51
st0             0              (raw 0x00000000000000000000)
st1             0              (raw 0x00000000000000000000)
st2             0              (raw 0x00000000000000000000)
st3             0              (raw 0x00000000000000000000)
st4             0              (raw 0x00000000000000000000)
st5             0              (raw 0x00000000000000000000)
st6             0              (raw 0x00000000000000000000)
st7             0              (raw 0x00000000000000000000)
fctrl           0x37f             895
fstat           0x0              0
ftag            0xffff             65535
fiseq           0x0              0
fioff           0x0              0
foseq           0x0              0
fooff           0x0              0
fop             0x0              0
-----
(gdb) x/5s 0xbffff5c0
0xbffff5c0:  "/bcc1"
0xbffff5c6:  "FAM-AB---NOMER123456789"
0xbffff5de:  "SHELL=/bin/bash"
0xbffff5ee:  "SESSION_MANAGER=local/ab-n:@tmp/.ICE-unix/1260,unix/ab-n:/tmp/.ICE-unix/1260"
0xbffff63c:  "QT_ACCESSIBILITY=1"

```

```

(gdb) n
input at 0x404024 FAM-AB---NOMER123456789
11     printf ("output at %p* %s\n", output, output);
(gdb) n
output at 0x404038* 789
12     printf("\n%s\n", output);
(gdb) n

789
13     }
(gdb) n
__libc_start_main (main=0x4011a9 <main>, argc=2, argv=0xbffff424, init=0x401270 <__libc_csu_init>,
fini=0x4012d0 <__libc_csu_fini>,
    rtdl_fini=0xb7fe6520 <_dl_fini>, stack_end=0xbffff41c) at ../csu/libc-start.c:342
342    ../csu/libc-start.c: No such file or directory.
(gdb) n
[Inferior 1 (process 2095) exited normally]
(gdb)

```

Выполнить анализ результата.

Использование KDbg иногда позволяет более наглядно представить результаты и анализ выполнения программы:

The screenshot shows the KDbg debugger interface with the following components:

- Menu Bar:** File, Edit, View, Execution, Breakpoint, Settings, Help.
- Toolbar:** Open Source, Find, Reload Source, Executable, Run, Step into, Step over, Step out.
- Source Code Window (bcc1.c):**

```

0x4011b6 push    %ebx
0x4011b7 push    %ecx
0x4011b8 call    0x4010b0 <_x86.get_pc_thunk.bx>
0x4011bd add     $0x2e43,%ebx
0x4011c3 mov     %ecx,%eax
+ 8  strcpy(output, "normal output");
+ 9  strcpy(input, argv[1]);
+ 10 printf ("input at %p %s\n", input, input);
+ 11 printf ("output at %p* %s\n", output, output);
+ 12 printf("\n%s\n", output);
+ 13 }
+ 14
+ 15

```
- Locals Window:**

Variable	Value
argc	2
argv	0xbffffa84
*0xbffffa84	0xbffffb14 "/home/fam_stud/Test/bcc1"
- Registers Window:**

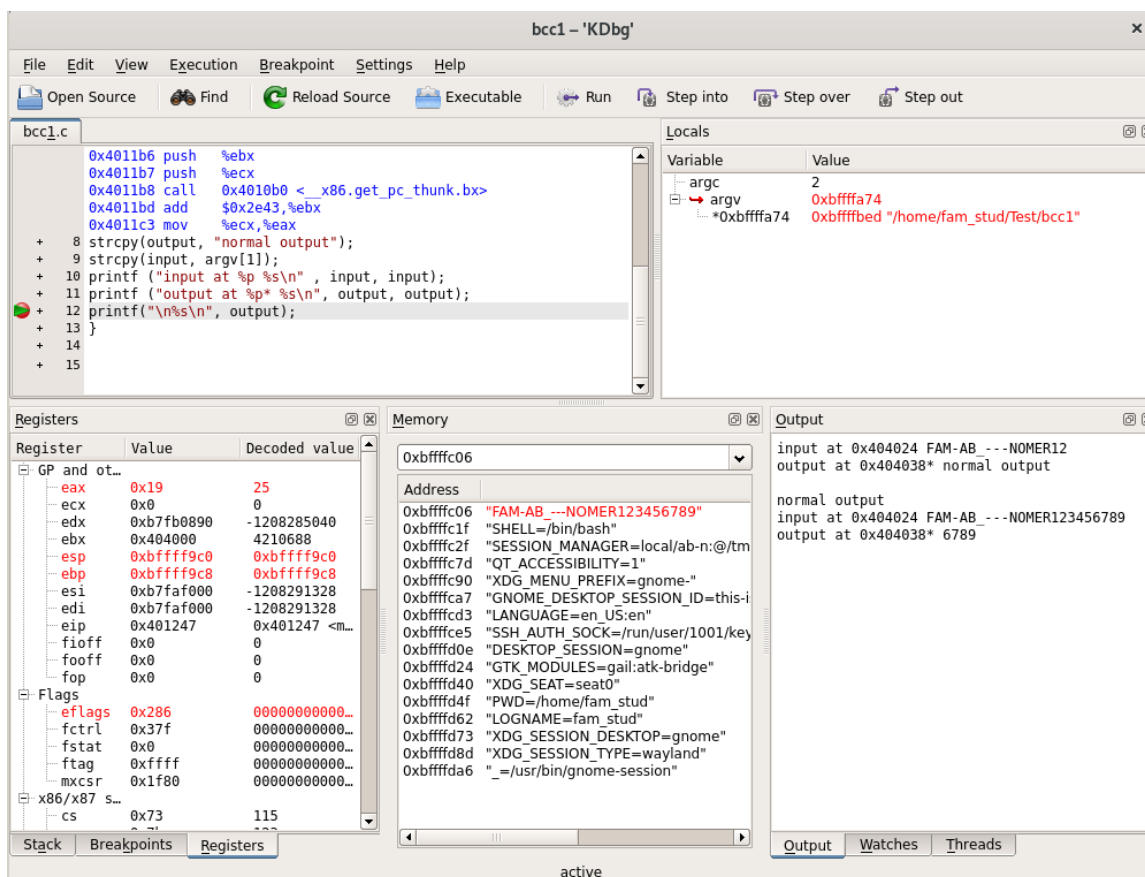
Register	Value	Decoded value
GP and ot...		
eax	0x22	34
ecx	0x0	0
edx	0xb7fb0890	-1208285040
ebx	0x404000	4210688
esp	0xbffff9d0	0xbffff9d0
ebp	0xbffff9d8	0xbffff9d8
esi	0xb7faf000	-1208291328
edi	0xb7faf000	-1208291328
eip	0x401247	0x401247 <m...
fioff	0x0	0
fooff	0x0	0
fop	0x0	0
Flags	0x282	000000000000...
eflags	0x282	000000000000...
fctrl	0x37f	000000000000...
fstat	0x0	000000000000...
ftag	0xffff	000000000000...
mxcscr	0x1f80	000000000000...
x86/x87 s...		
cs	0x73	115
- Memory Window:**

Address	Value
0xbffffc06	"t/bcc1"
0xbffffc0d	"FAM-AB ---NOMER12"
0xbffffc1f	"SHELL=/bin/bash"
0xbffffc2f	"SESSION_MANAGER=local/ab-n:@/tm"
0xbffffc7d	"QT_ACCESSIBILITY=1"
0xbffffc90	"XDG_MENU_PREFIX=gnome-"
0xbffffca7	"GNOME_DESKTOP_SESSION_ID=this-i"
0xbffffcd3	"LANGUAGE=en_US:en"
0xbffffce5	"SSH_AUTH_SOCK=/run/user/1001/key"
0xbffffd0e	"DESKTOP_SESSION=gnome"
0xbffffd24	"GTK_MODULES=gail:atk-bridge"
0xbffffd40	"XDG_SEAT=seat0"
0xbffffd4f	"PWD=/home/fam_stud"
0xbffffd62	"LOGNAME=fam_stud"
0xbffffd73	"XDG_SESSION_DESKTOP=gnome"
0xbffffd8d	"XDG_SESSION_TYPE=wayland"
- Output Window:**

```

input at 0x404024 FAM-AB ---NOMER12
output at 0x404038* normal output

```



Сравнить полученные результаты 6.1 и 6.2

Дополнительно могут быть представлены результаты рассмотрения уязвимостей из источников [Лит. - 1] и [Доп. - 1,2].

По результатам выполнения сформулировать выводы о наличии уязвимостей и возможности реализации атак.

Литература:

1. Фостер Дж., Лю В. Разработка средств безопасности и эксплойтов / Пер. с англ. — М. : Издательство «Русская Редакция» ; СПб. : Питер, 2007. — 432 стр. : ил.
2. Роббинс А. Linux: программирование в примерах. Пер с англ. - М.: КУДИЦ-ОБРАЗ, 2005. - 656 с.
3. Лав Р. Linux. Системное программирование. 2-е изд. — СПб.: Питер, 2014. — 448 с.: ил. — (Серия «Бестселлеры O'Reilly»).
4. Richard Stallman, Roland Pesch, Stan Shebs, et al. Debugging with gdb. Tenth Edition.

Дополнительно:

1. Elisan Christopher C. Advanced Malware Analysis. 2015 by McGraw-Hill Education. - 545 p.
2. Michael Sikorski and Andrew Honig. PRACTICAL MALWARE ANALYSIS. No Starch Press, Inc. 2012 -802 p.
3. Kaiwan N Billimoria. Hands-On System Programming with Linux 2012