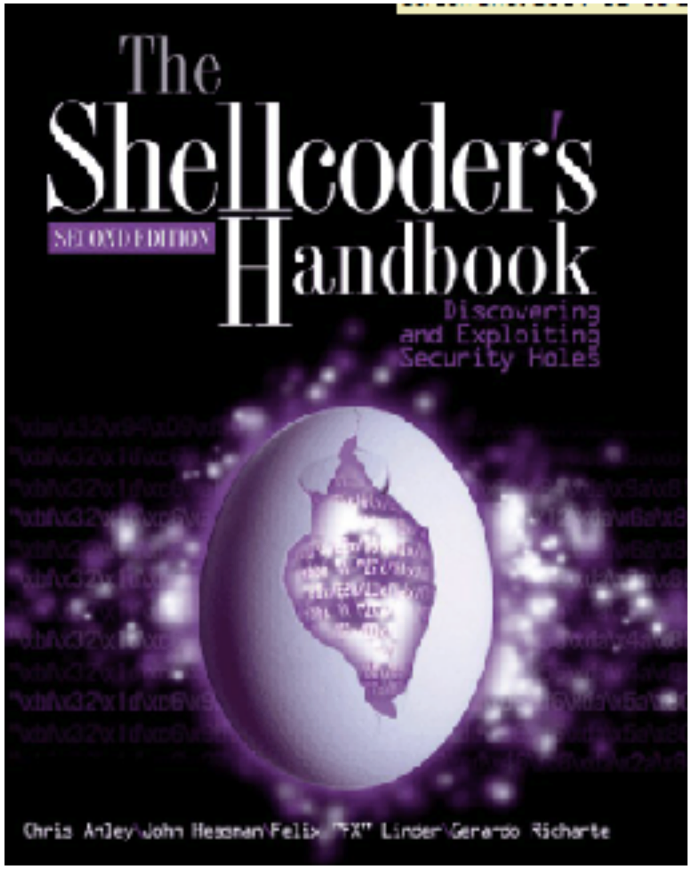
CNIT 127: Exploit Development

Ch 2: Stack Overflows in Linux


The Shellcoder's Handbook
SECOND EDITION
Discovering and Exploiting Security Holes
Chris Anley\John Heasman\Felix "FX" Linder\Gerardo Richarte

Updated 8-28-19

Topics

- Buffers in C
- Information Disclosure
- gdb: Gnu Debugger
- Segmentation Fault
- The Stack
- Functions and the Stack
- Stack Buffer Overflow

Stack-Based Buffer Overflows

- Most popular and best understood exploitation method
- Aleph One's "Smashing the Stack for Fun and Profit" (1996)
  - Link Ch 2a
- Buffer
  - A limited, contiguously allocated set of memory
  - In C, usually an *array*

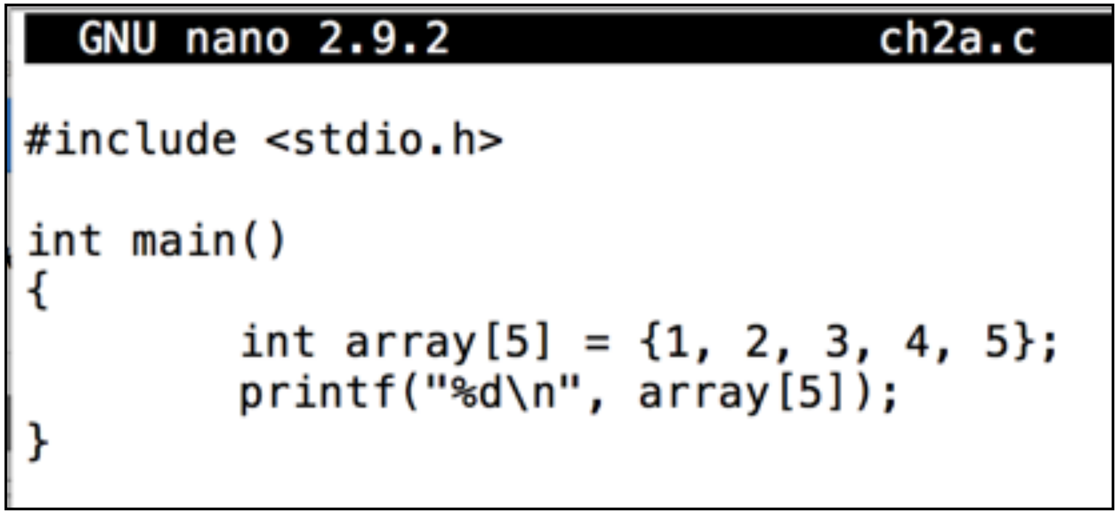
Exploit A: Information Disclosure

C and C++ Lack Bounds-Checking

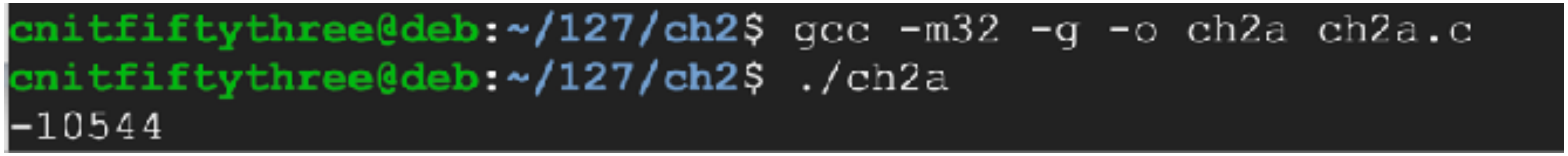
- It is the programmer's responsibility to ensure that array indices remain in the valid range

```
#include <stdio.h>

int main()
{
        int array[5] = {1, 2, 3, 4, 5};
        printf("%d\n", array[5]);
}
```

Reading Past End of Array



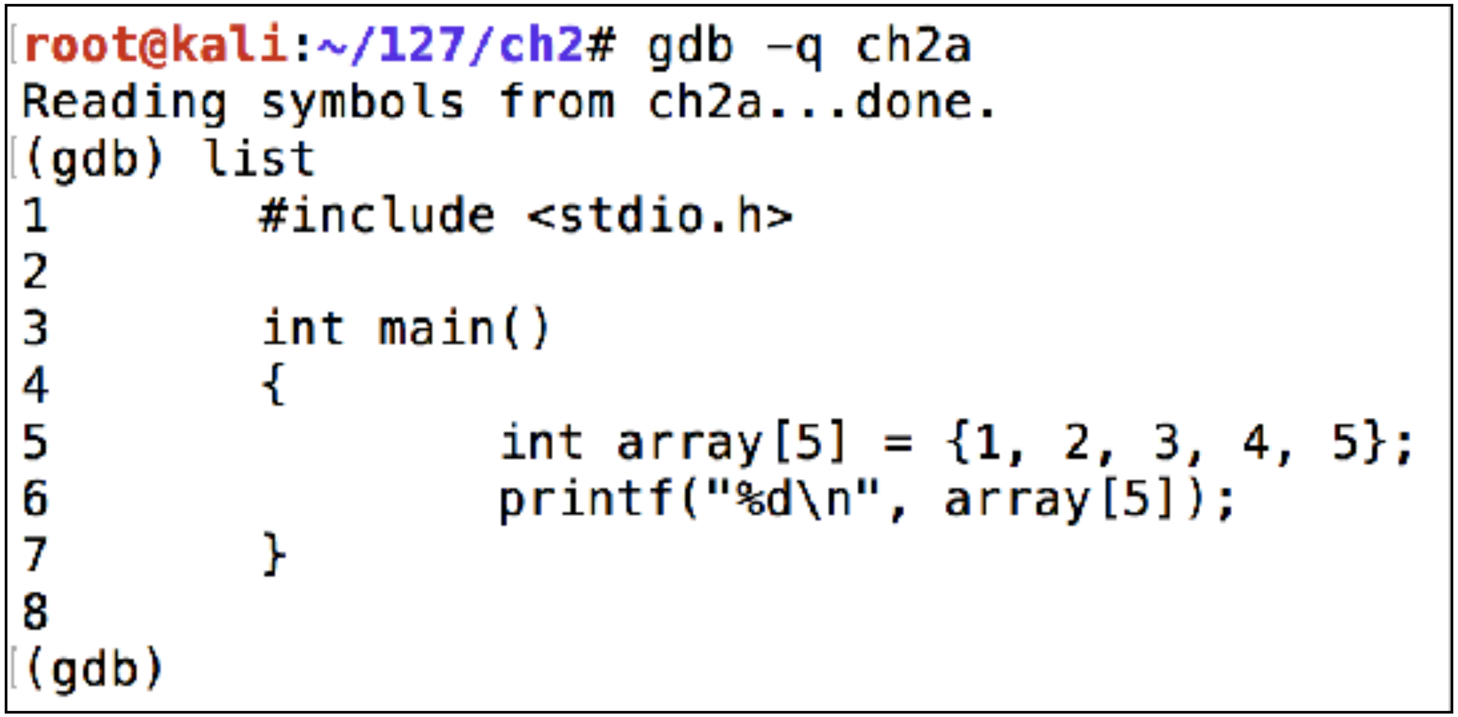
```
  GNU nano 2.9.2                    ch2a.c

#include <stdio.h>

int main()
{
        int array[5] = {1, 2, 3, 4, 5};
        printf("%d\n", array[5]);
}
```



```
cnitfiftythree@deb:~/127/ch2$ gcc -m32 -g -o ch2a ch2a.c
cnitfiftythree@deb:~/127/ch2$ ./ch2a
-10544
```

- We can **read** data that we shouldn't be seeing
- *Information disclosure vulnerabilty*

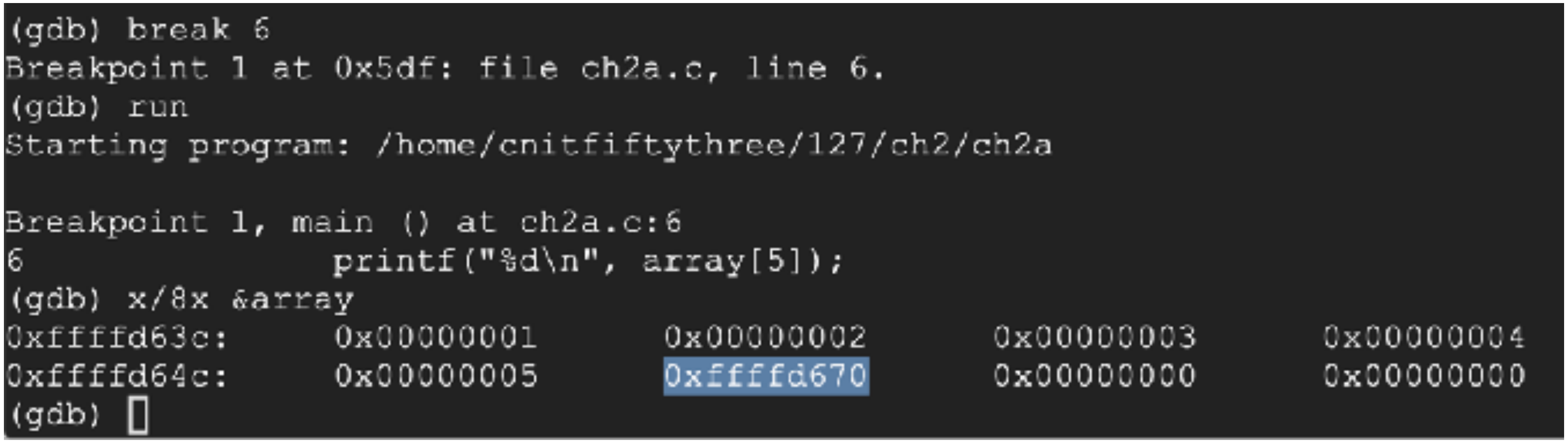
Using gdb (GNU Debugger)



```
root@kali:~/127/ch2# gdb -q ch2a
Reading symbols from ch2a...done.
(gdb) list
1       #include <stdio.h>
2
3       int main()
4       {
5               int array[5] = {1, 2, 3, 4, 5};
6               printf("%d\n", array[5]);
7       }
8
(gdb)
```

- **Source code debugging**
- Because we compiled with **gcc -g**

Using gdb (GNU Debugger)



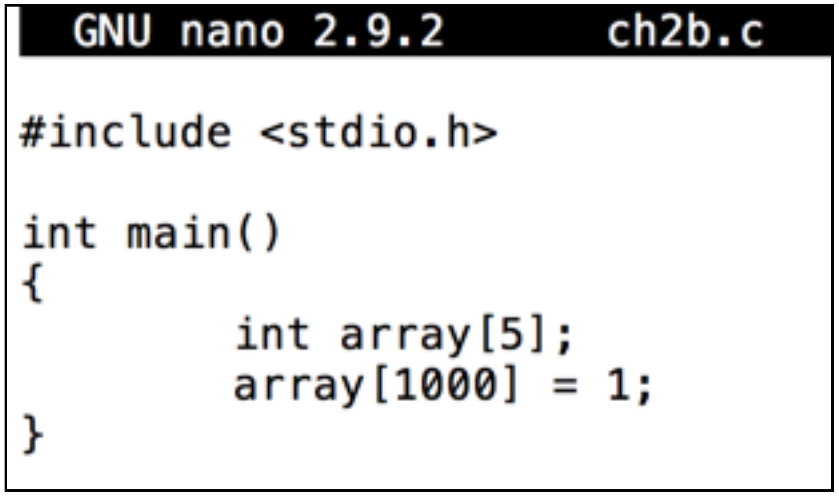
```
(gdb) break 6
Breakpoint 1 at 0x5df: file ch2a.c, line 6.
(gdb) run
Starting program: /home/cnitfiftythree/127/ch2/ch2a

Breakpoint 1, main () at ch2a.c:6
6               printf("%d\n", array[5]);
(gdb) x/8x &array
0xffffd63c:     0x00000001      0x00000002      0x00000003      0x00000004
0xffffd64c:     0x00000005      0xffffd670      0x00000000      0x00000000
(gdb) 
```

- gdb commands

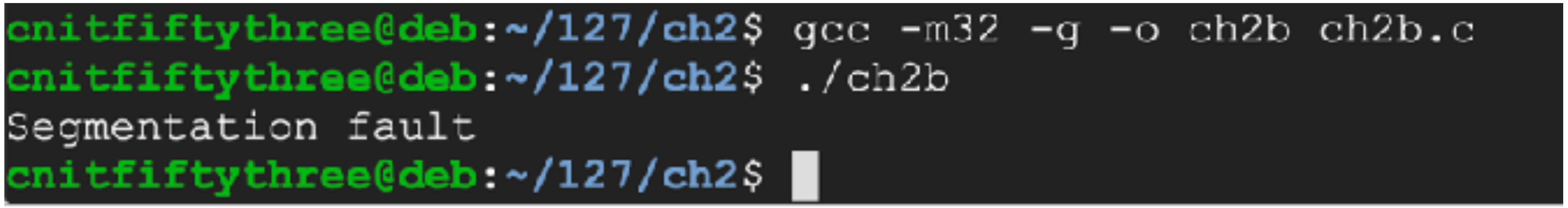
list	*show source code*
run	*execute program*
break	*insert breakpoint*
x	*examine memory*

Exploit B: Denial of Service

Writing Past End of Array



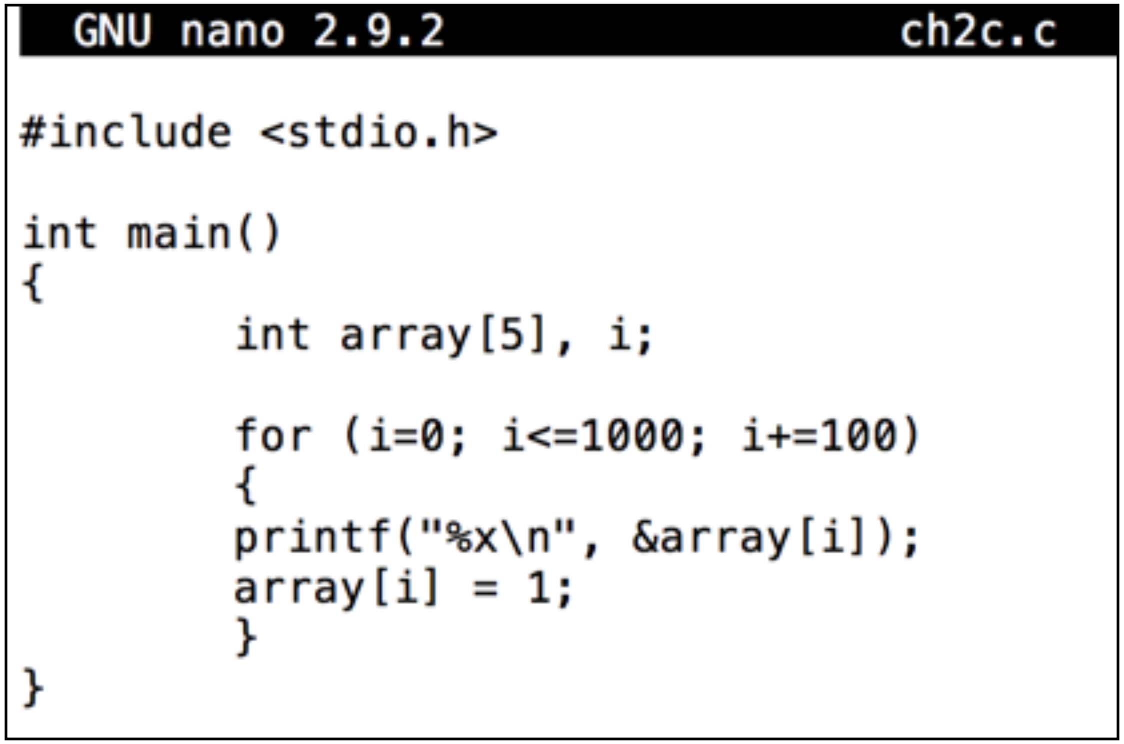
```
  GNU nano 2.9.2          ch2b.c

#include <stdio.h>

int main()
{
        int array[5];
        array[1000] = 1;
}
```



```
cnitfiftythree@deb:~/127/ch2$ gcc -m32 -g -o ch2b ch2b.c
cnitfiftythree@deb:~/127/ch2$ ./ch2b
Segmentation fault
cnitfiftythree@deb:~/127/ch2$ 
```

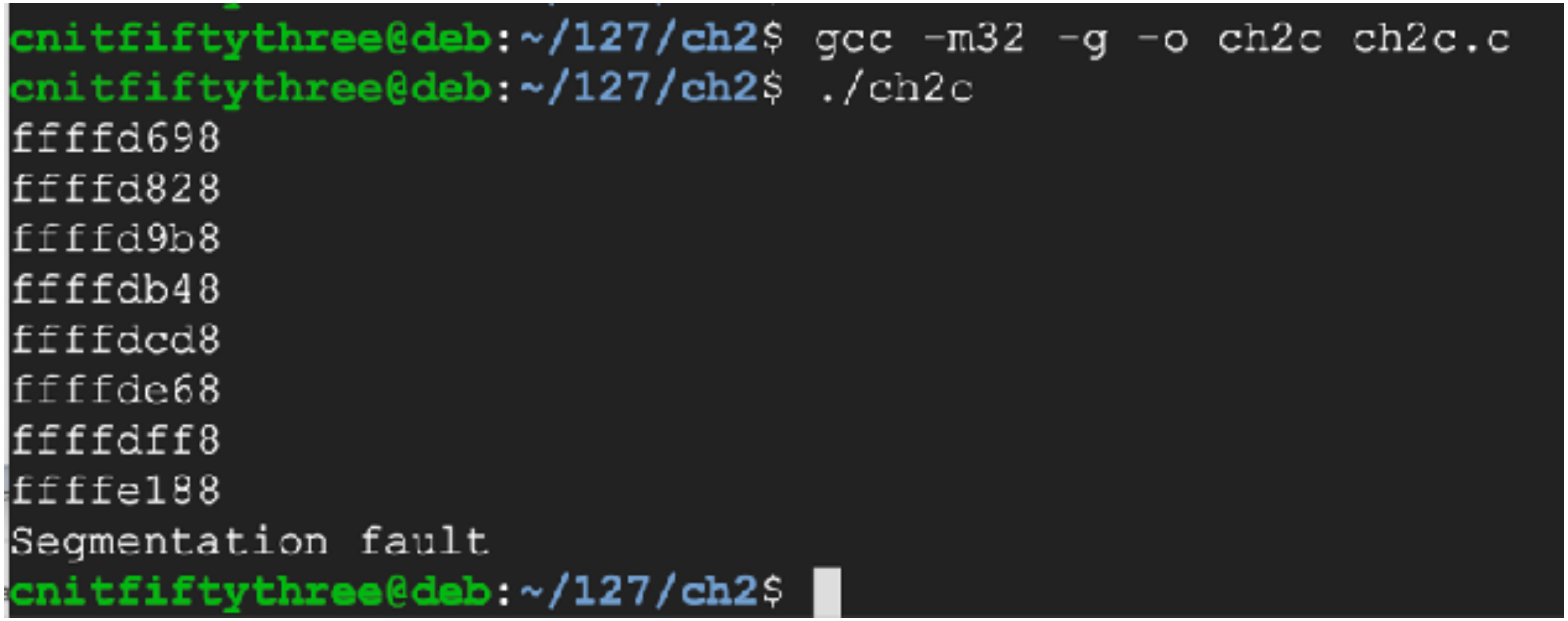
- Program has crashed
- *Denial of service*

Print Out More Information


```
  GNU nano 2.9.2                    ch2c.c

#include <stdio.h>

int main()
{
        int array[5], i;

        for (i=0; i<=1000; i+=100)
        {
        printf("%x\n", &array[i]);
        array[i] = 1;
        }
}
```

- printf uses a **format string**
- **%x** means print in hexadecimal

Segmentation Fault

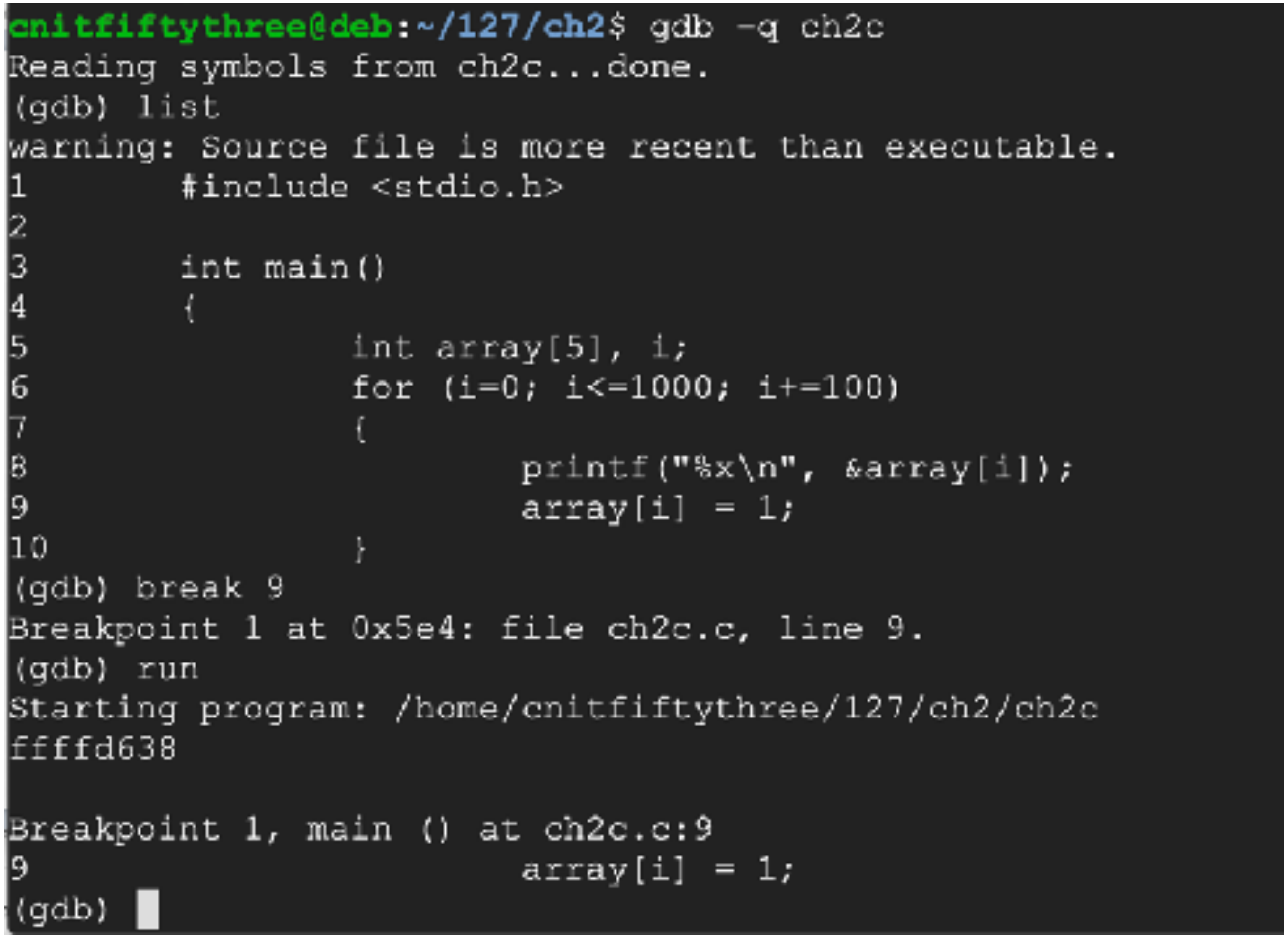


```
cnitfiftythree@deb:~/127/ch2$ gcc -m32 -g -o ch2c ch2c.c
cnitfiftythree@deb:~/127/ch2$ ./ch2c
ffffd698
ffffd828
ffffd9b8
ffffdb48
ffffdcd8
ffffde68
ffffdff8
ffffe188
Segmentation fault
cnitfiftythree@deb:~/127/ch2$
```

- Cannot write to address ffffe188

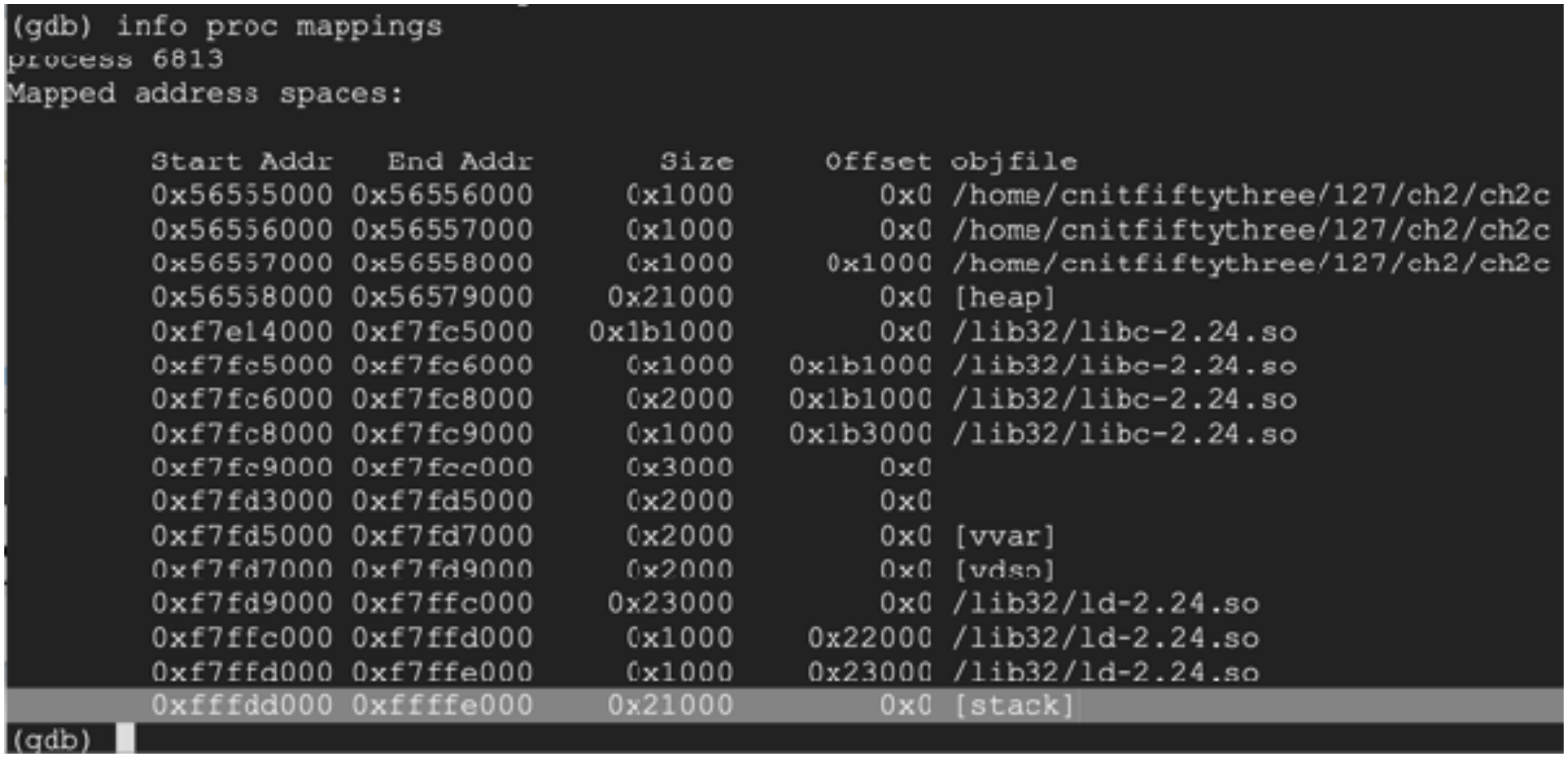
Debug

Insert breakpoint and run



```
cnitfiftythree@deb:~/127/ch2$ gdb -q ch2c
Reading symbols from ch2c...done.
(gdb) list
warning: Source file is more recent than executable.
1       #include <stdio.h>
2
3       int main()
4       {
5               int array[5], i;
6               for (i=0; i<=1000; i+=100)
7               {
8                       printf("%x\n", &array[i]);
9                       array[i] = 1;
10              }
(gdb) break 9
Breakpoint 1 at 0x5e4: file ch2c.c, line 9.
(gdb) run
Starting program: /home/cnitfiftythree/127/ch2/ch2c
ffffd638

Breakpoint 1, main () at ch2c.c:9
9                       array[i] = 1;
(gdb) 
```

Memory Map



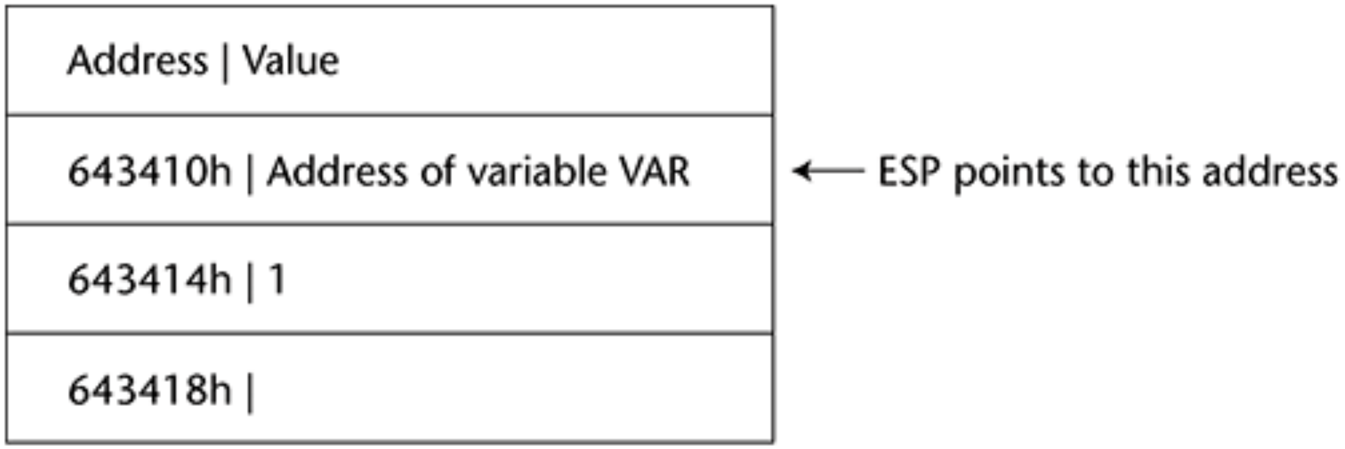
```
(gdb) info proc mappings
process 6813
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
        0x56555000 0x56556000     0x1000        0x0 /home/cnitfiftythree/127/ch2/ch2c
        0x56556000 0x56557000     0x1000        0x0 /home/cnitfiftythree/127/ch2/ch2c
        0x56557000 0x56558000     0x1000     0x1000 /home/cnitfiftythree/127/ch2/ch2c
        0x56558000 0x56579000    0x21000        0x0 [heap]
        0xf7e14000 0xf7fc5000   0x1b1000        0x0 /lib32/libc-2.24.so
        0xf7fc5000 0xf7fc6000     0x1000   0x1b1000 /lib32/libc-2.24.so
        0xf7fc6000 0xf7fc8000     0x2000   0x1b1000 /lib32/libc-2.24.so
        0xf7fc8000 0xf7fc9000     0x1000   0x1b3000 /lib32/libc-2.24.so
        0xf7fc9000 0xf7fcc000     0x3000        0x0
        0xf7fd3000 0xf7fd5000     0x2000        0x0
        0xf7fd5000 0xf7fd7000     0x2000        0x0 [vvar]
        0xf7fd7000 0xf7fd9000     0x2000        0x0 [vdso]
        0xf7fd9000 0xf7ffc000    0x23000        0x0 /lib32/ld-2.24.so
        0xf7ffc000 0xf7ffd000     0x1000    0x22000 /lib32/ld-2.24.so
        0xf7ffd000 0xf7ffe000     0x1000    0x23000 /lib32/ld-2.24.so
        0xfffdd000 0xffffe000    0x21000        0x0 [stack]
(gdb)
```

- **Stack** ends at **0xffffe000**
- Trying to write past this address caused a segmentation fault

The Stack

LIFO (Last-In, First-Out)

- ESP (Extended Stack Pointer) register points to the top of the stack
- PUSH puts items on the stack
  - push 1
  - push addr var

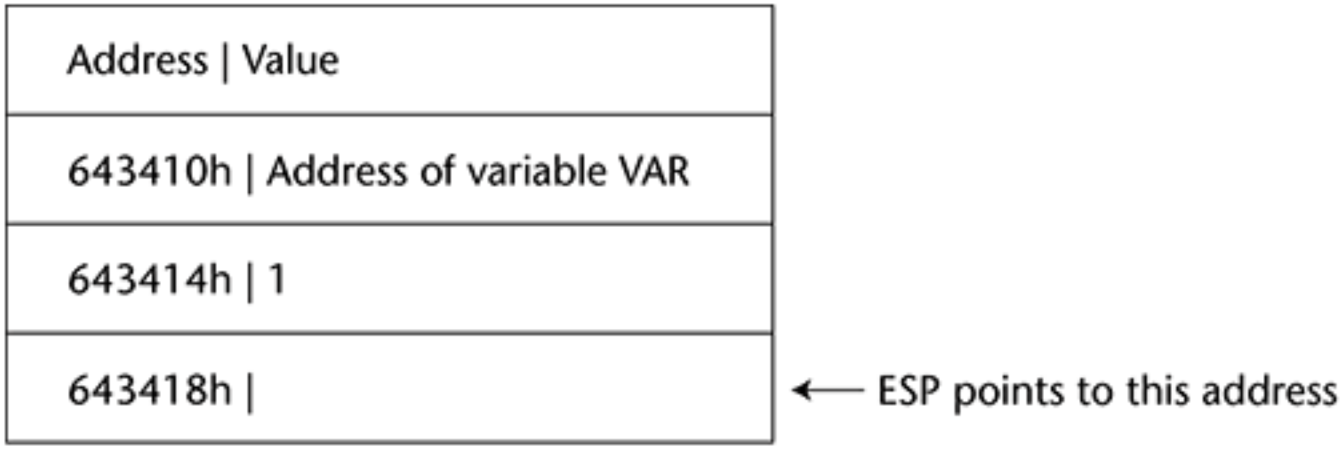


Address | Value
643410h | Address of variable VAR
643414h | 1
643418h |

← ESP points to this address

Stack

- POP takes items off the stack
  - pop eax
  - pop ebx



Address | Value
643410h | Address of variable VAR
643414h | 1
643418h |

EBP (Extended Base Pointer)

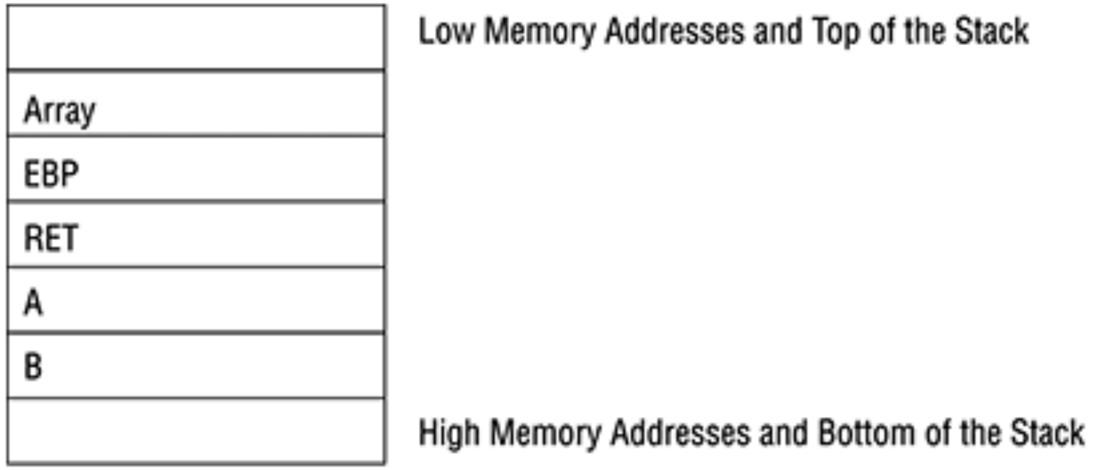
- EBP is typically used for calculated addresses on the stack
  - mov eax, [ebp+10h]
- Copies the data 16 bytes down the stack into the EAX register

Functions and the Stack

Purpose

- The stack's primary purpose is to make the use of functions more efficient
- When a function is called, these things occur:
  - Calling routine stops processing its instructions
  - Saves its current state
  - Transfers control to the function
  - Function processes its instructions
  - Function exits
  - State of the calling function is restored
  - Calling routine's execution resumes

Figure 2-3: Visual representation of the stack after a function has been called


Low Memory Addresses and Top of the Stack
Array
EBP
RET
A
B
High Memory Addresses and Bottom of the Stack

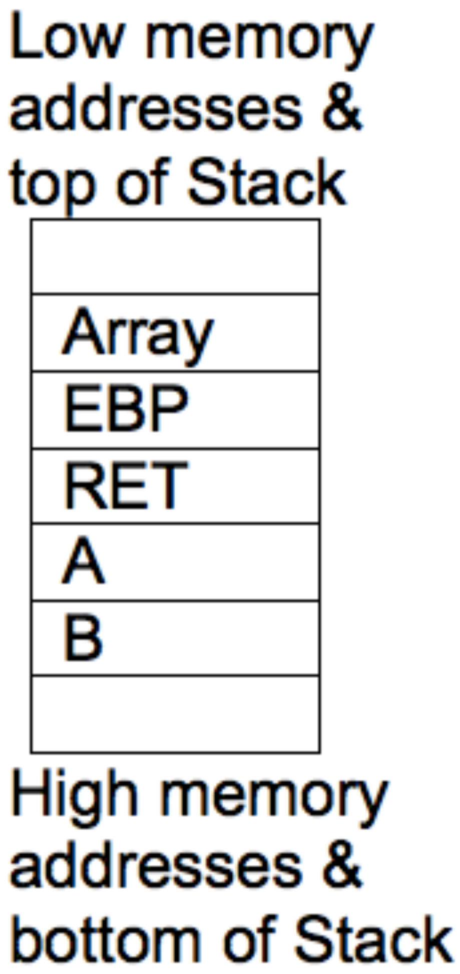
Functions and the Stack

- Primary purpose of the stack
  - To make functions more efficient
- When a function is called
  - Push function's **arguments** onto the stack
  - Call function, which pushes the return address **RET** onto the stack, which is the **EIP** at the time the function is called

Functions and the Stack

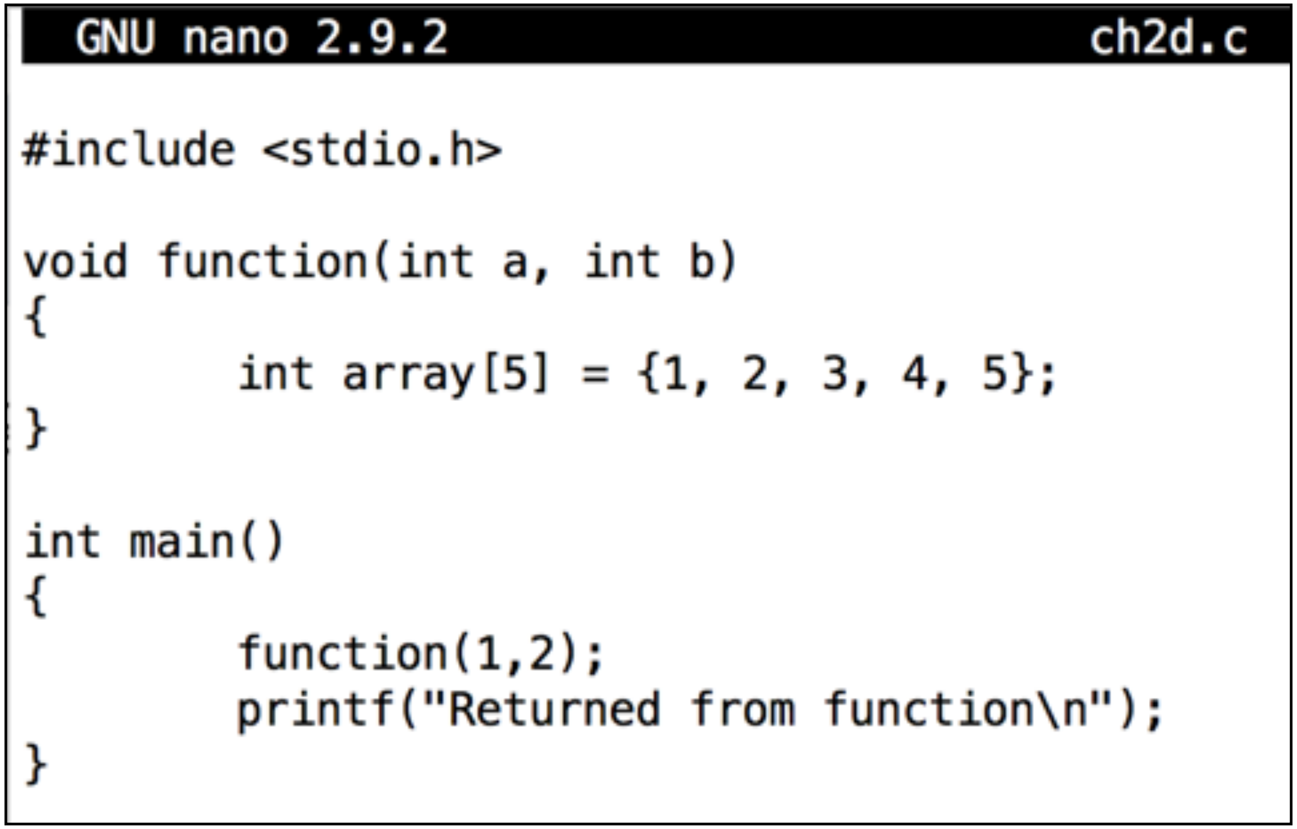
- Before function starts, a **prolog** executes, pushing **EBP** onto the stack
- It then copies **ESP** into **EBP**
- Calculates size of local variables
- Reserves that space on the stack, by subtracting the size from **ESP**
- Pushes local variables onto stack

Functions and the Stack

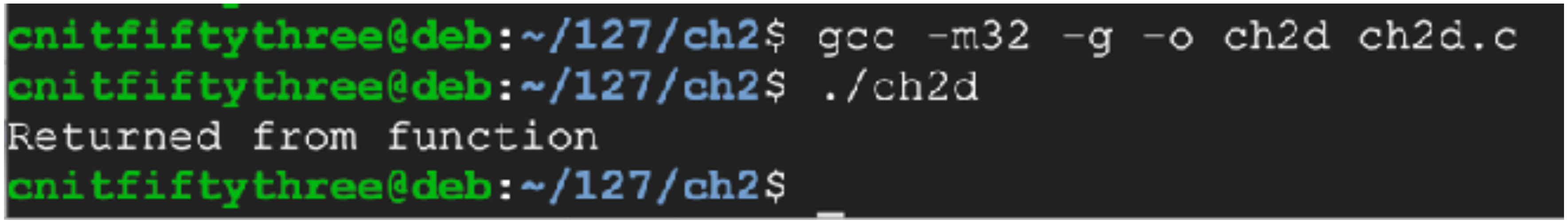

Low memory addresses & top of Stack
Array
EBP
RET
A
B
High memory addresses & bottom of Stack

```
#include <stdio.h>


void function(int a, int b)
{
   int array[5];
}


main()
{
   function(1,2);
   printf("This is where the
   return address points\n");
}
```

Example of a Function

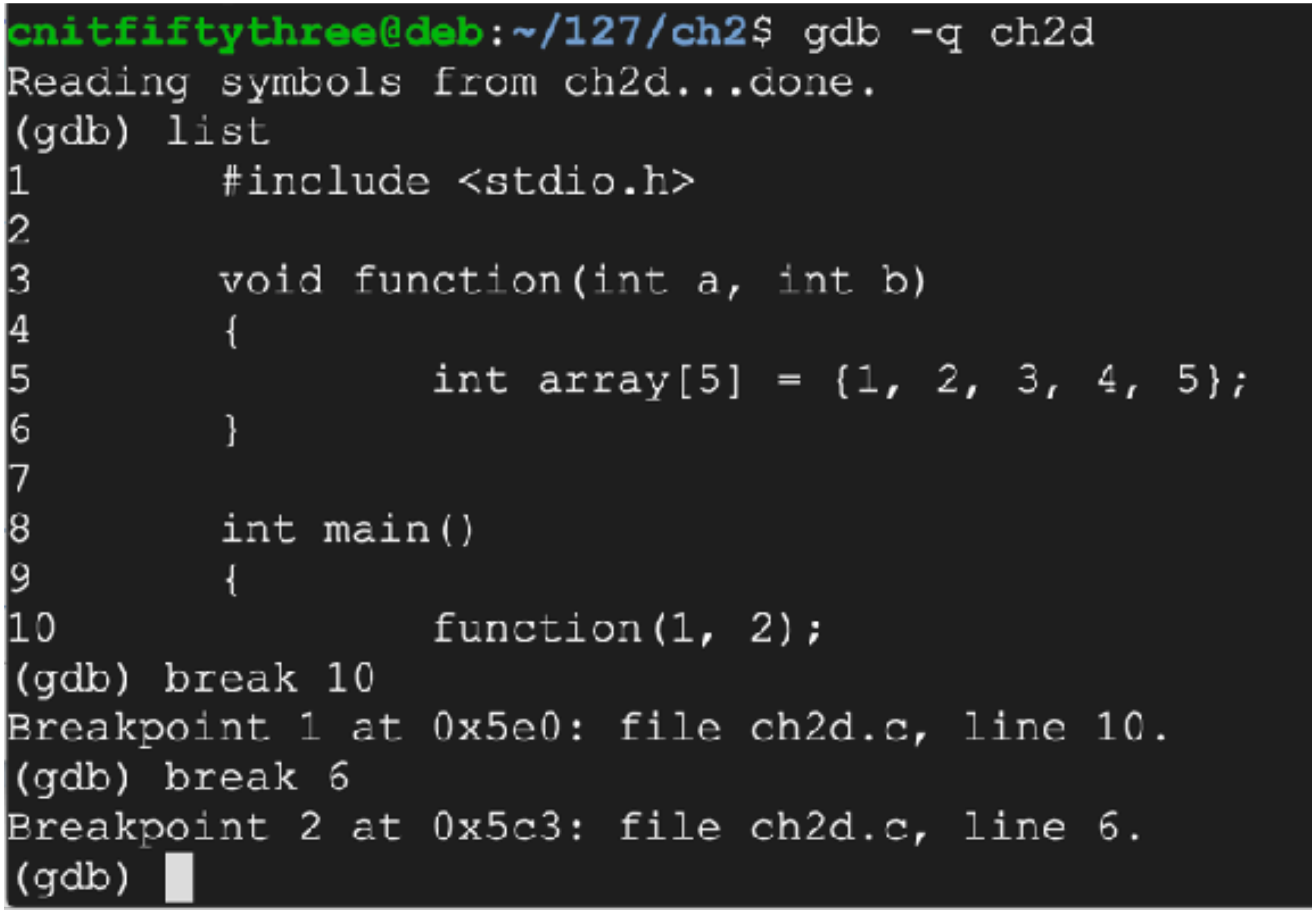


```
 GNU nano 2.9.2                                    ch2d.c

#include <stdio.h>

void function(int a, int b)
{
        int array[5] = {1, 2, 3, 4, 5};
}

int main()
{
        function(1,2);
        printf("Returned from function\n");
}
```



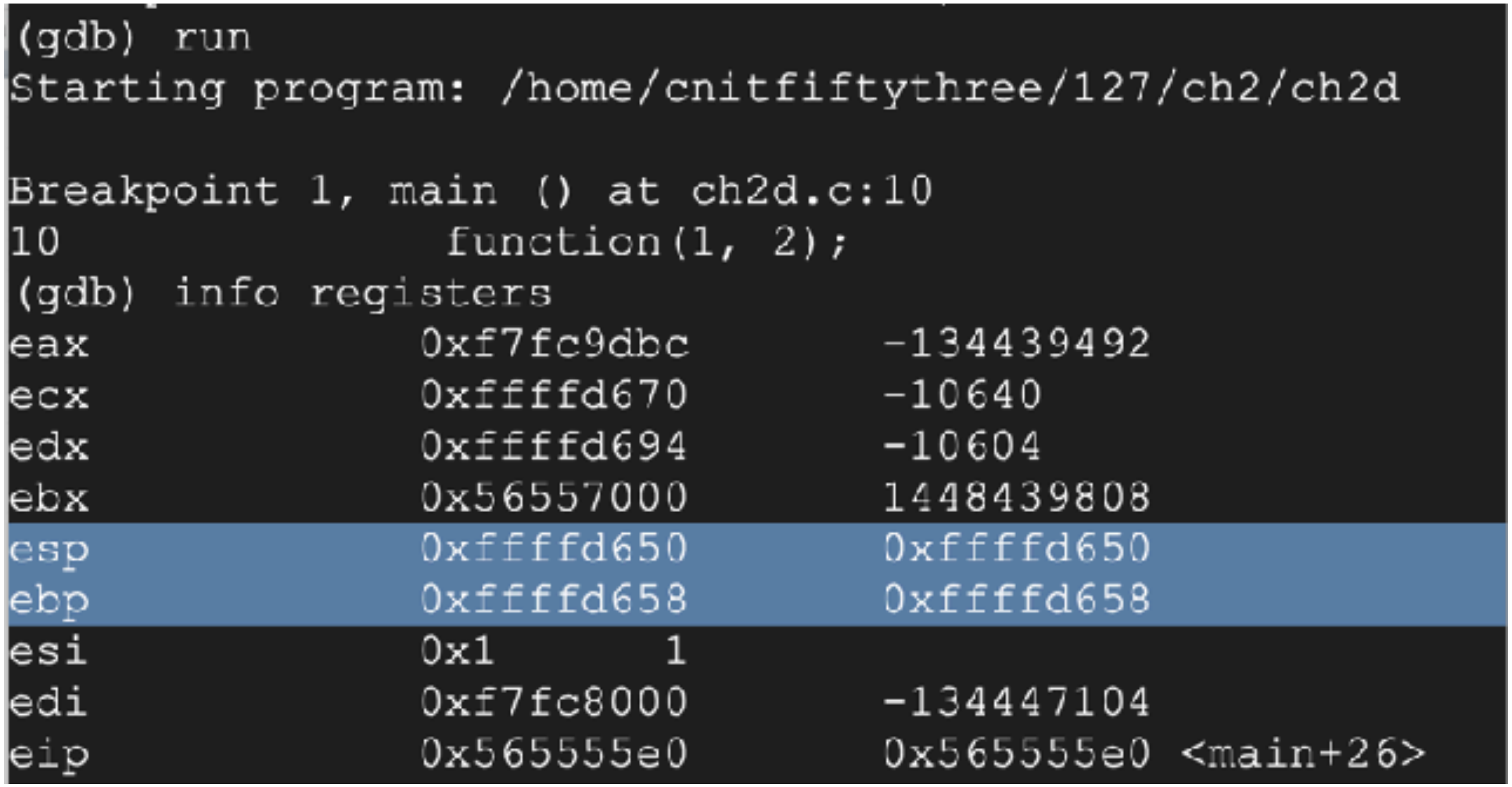
```
cnitfiftythree@deb:~/127/ch2$ gcc -m32 -g -o ch2d ch2d.c
cnitfiftythree@deb:~/127/ch2$ ./ch2d
Returned from function
cnitfiftythree@deb:~/127/ch2$
```

Debug and Set Breakpoints



```
cnitfiftythree@deb:~/127/ch2$ gdb -q ch2d
Reading symbols from ch2d...done.
(gdb) list
1       #include <stdio.h>
2
3       void function(int a, int b)
4       {
5               int array[5] = {1, 2, 3, 4, 5};
6       }
7
8       int main()
9       {
10              function(1, 2);
(gdb) break 10
Breakpoint 1 at 0x5e0: file ch2d.c, line 10.
(gdb) break 6
Breakpoint 2 at 0x5c3: file ch2d.c, line 6.
(gdb) 
```

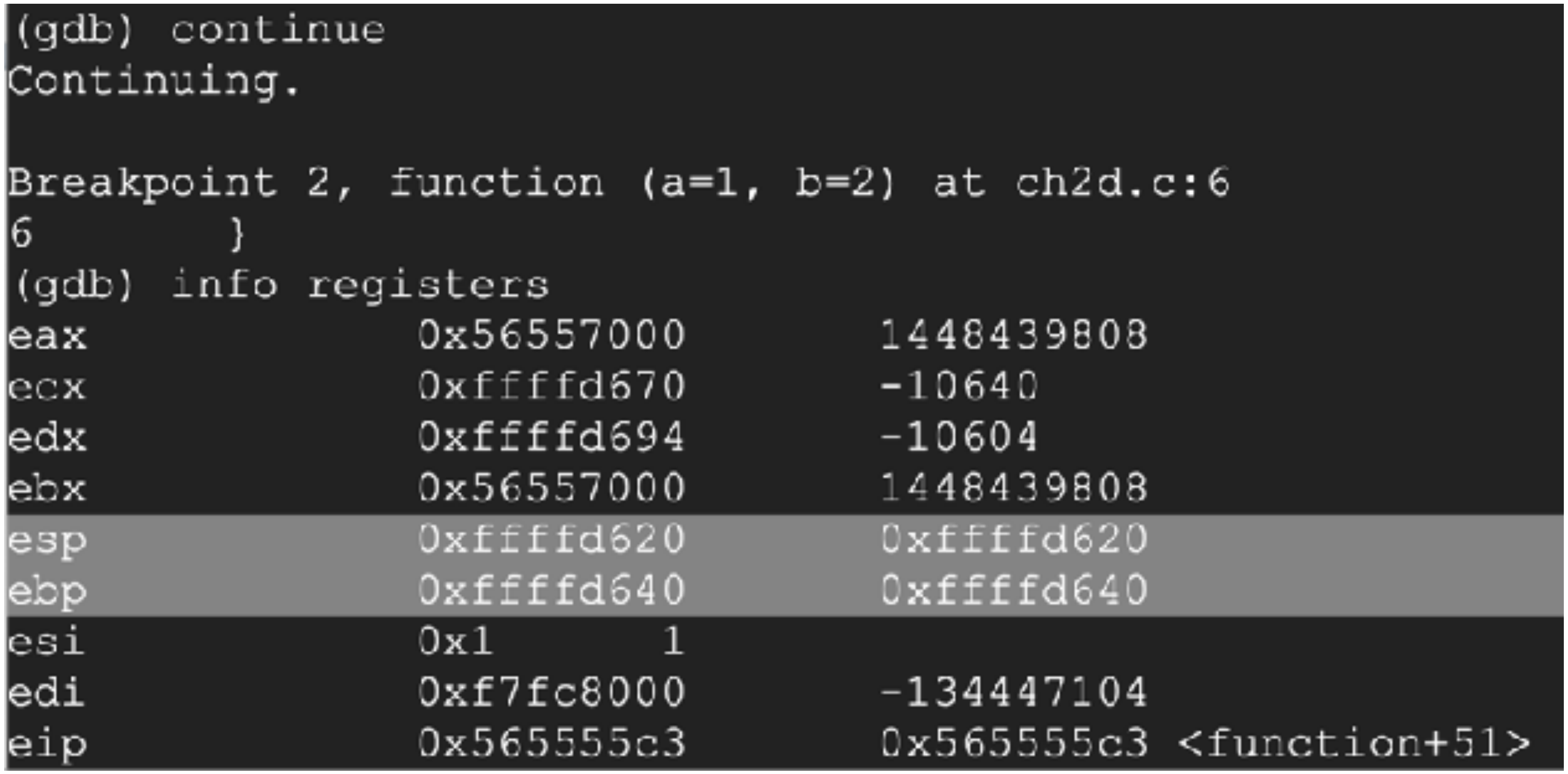
In main()



```
(gdb) run
Starting program: /home/cnitfiftythree/127/ch2/ch2d

Breakpoint 1, main () at ch2d.c:10
10              function(1, 2);
(gdb) info registers
eax            0xf7fc9dbc       -134439492
ecx            0xffffd670       -10640
edx            0xffffd694       -10604
ebx            0x56557000       1448439808
esp            0xffffd650       0xffffd650
ebp            0xffffd658       0xffffd658
esi            0x1      1
edi            0xf7fc8000       -134447104
eip            0x565555e0       0x565555e0 <main+26>
```

Stack frame goes from ebp to esp

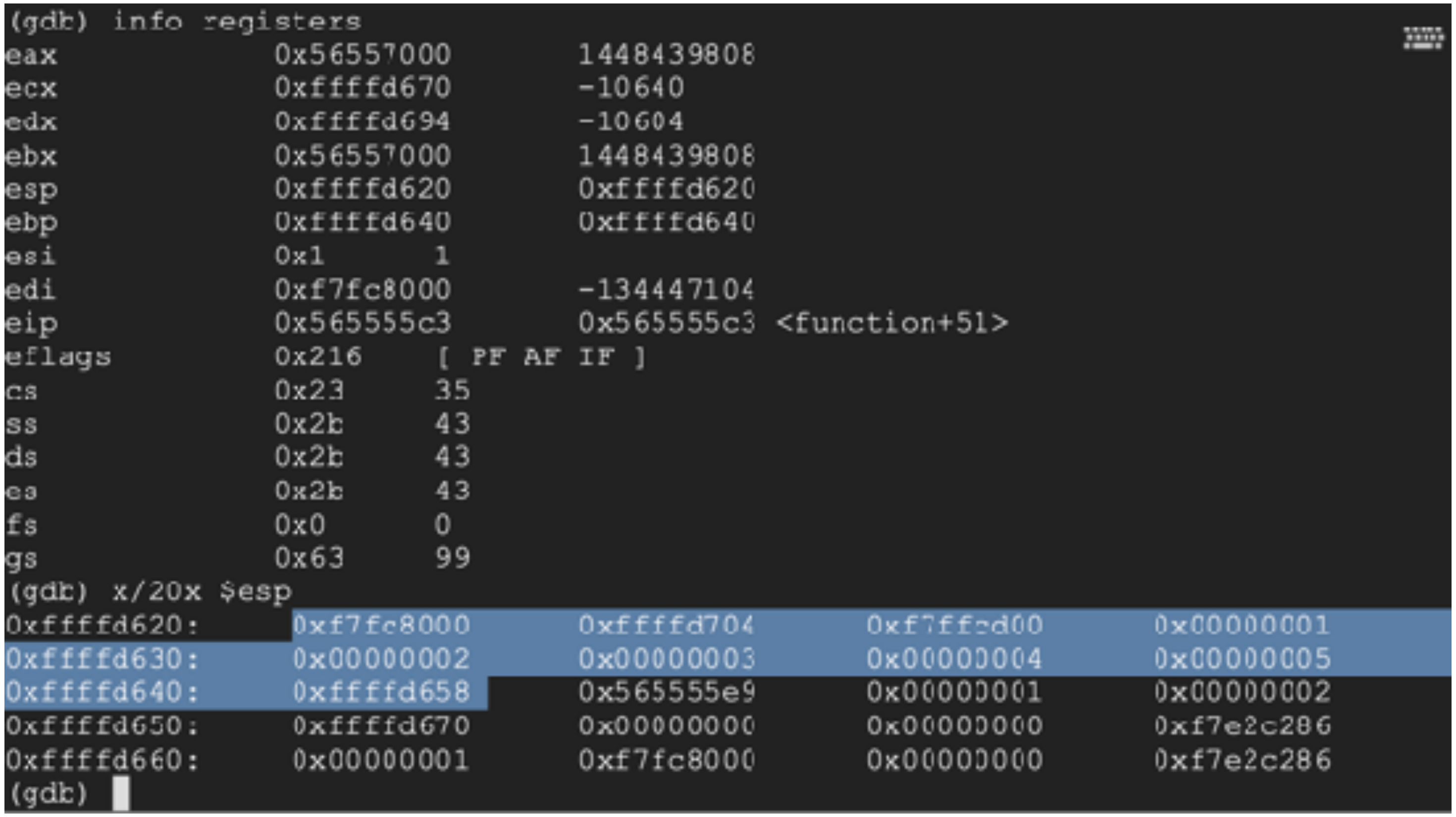
In function()



```
(gdb) continue
Continuing.

Breakpoint 2, function (a=1, b=2) at ch2d.c:6
6         }
(gdb) info registers
eax            0x56557000       1448439808
ecx            0xffffd670       -10640
edx            0xffffd694       -10604
ebx            0x56557000       1448439808
esp            0xffffd620       0xffffd620
ebp            0xffffd640       0xffffd640
esi            0x1      1
edi            0xf7fc8000       -134447104
eip            0x565555c3       0x565555c3 <function+51>
```

Stack frame goes from ebp to esp

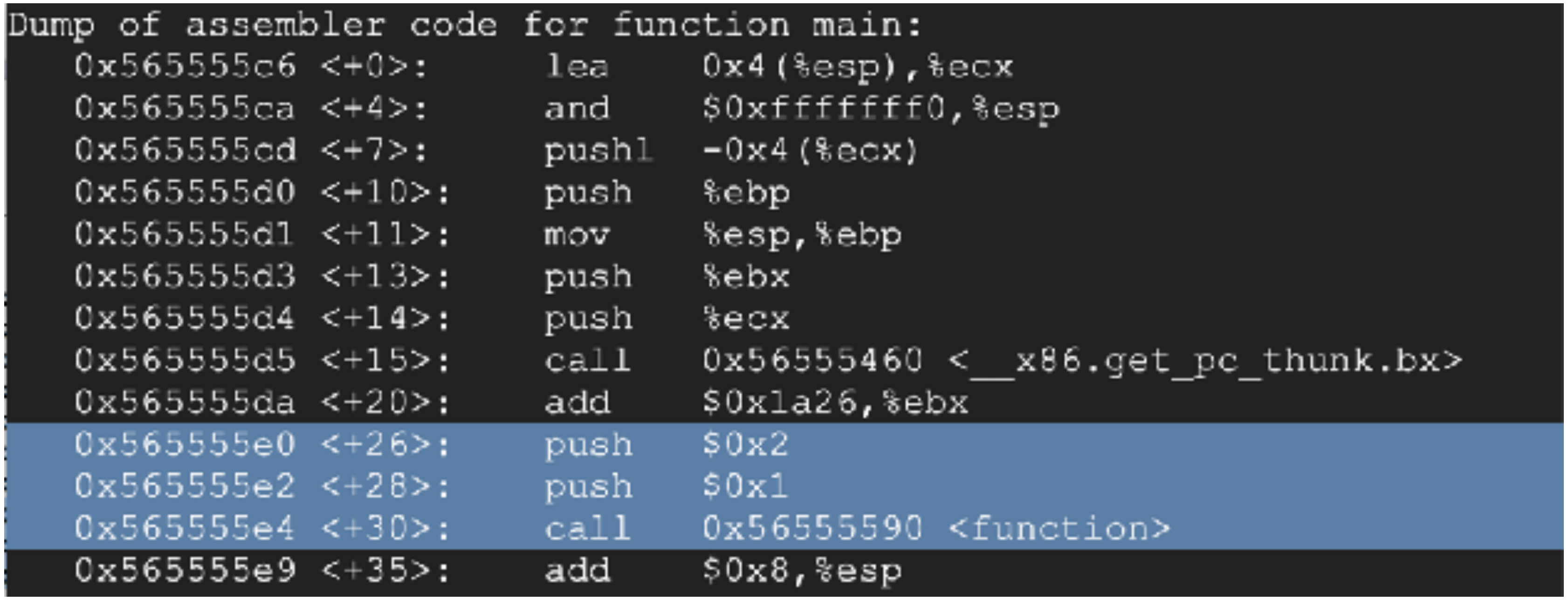
Examine the Stack Frame



```
(gdb) info registers
eax            0x56557000       1448439808
ecx            0xffffd670       -10640
edx            0xffffd694       -10604
ebx            0x56557000       1448439808
esp            0xffffd620       0xffffd620
ebp            0xffffd640       0xffffd640
esi            0x1     1
edi            0xf7fc8000       -134447104
eip            0x565555c3       0x565555c3 <function+51>
eflags         0x216    [ PF AF IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x63     99
(gdb) x/20x $esp
0xffffd620:     0xf7fc8000      0xffffd704      0xf7ffcd00      0x00000001
0xffffd630:     0x00000002      0x00000003      0x00000004      0x00000005
0xffffd640:     0xffffd658      0x565555e9      0x00000001      0x00000002
0xffffd650:     0xffffd670      0x00000000      0x00000000      0xf7e2c286
0xffffd660:     0x00000001      0xf7fc8000      0x00000000      0xf7e2c286
(gdb)
```

- Highlighted region is the **stack frame** of function()
- Below it is the stack frame of main()

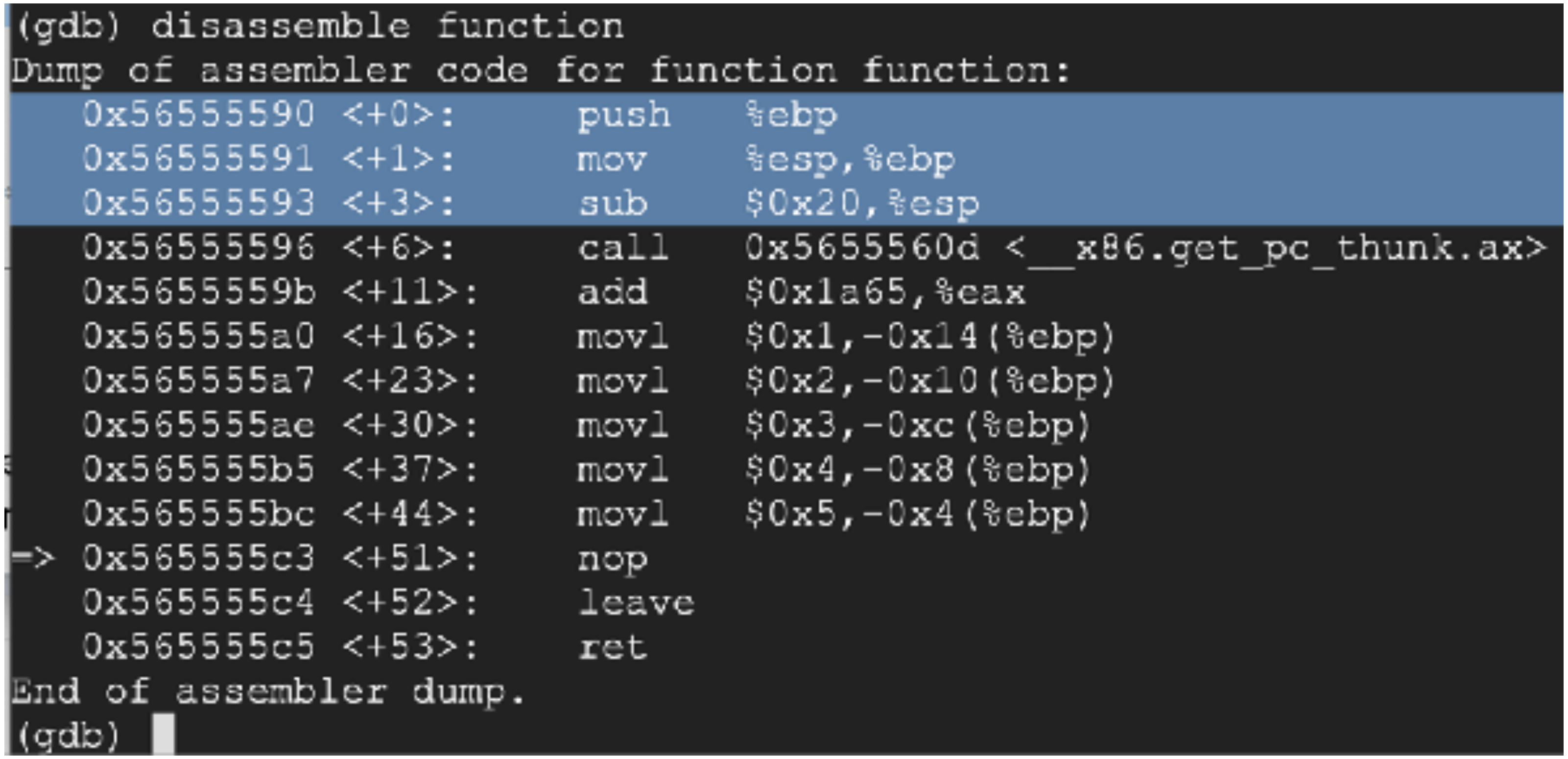
Disassemble Main



```
Dump of assembler code for function main:
   0x565555c6 <+0>:     lea    0x4(%esp),%ecx
   0x565555ca <+4>:     and    $0xfffffff0,%esp
   0x565555cd <+7>:     pushl  -0x4(%ecx)
   0x565555d0 <+10>:    push   %ebp
   0x565555d1 <+11>:    mov    %esp,%ebp
   0x565555d3 <+13>:    push   %ebx
   0x565555d4 <+14>:    push   %ecx
   0x565555d5 <+15>:    call   0x56555460 <__x86.get_pc_thunk.bx>
   0x565555da <+20>:    add    $0x1a26,%ebx
   0x565555e0 <+26>:    push   $0x2
   0x565555e2 <+28>:    push   $0x1
   0x565555e4 <+30>:    call   0x56555590 <function>
   0x565555e9 <+35>:    add    $0x8,%esp
```

- To call a function:
  - **push** arguments onto the stack
  - **call** the function

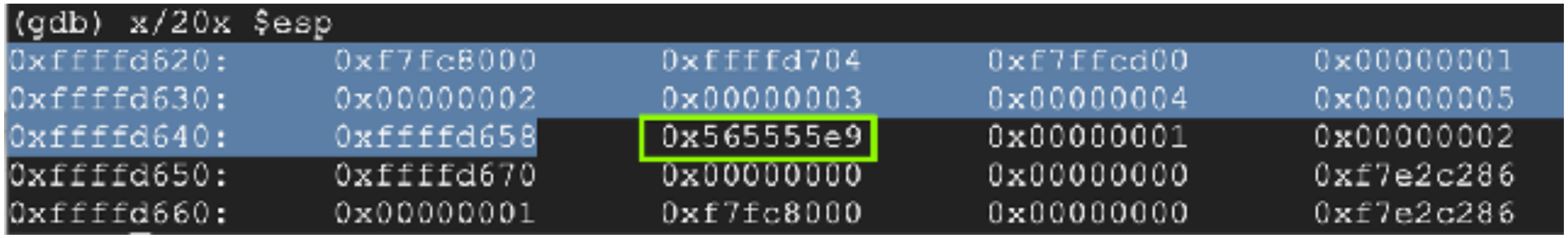
Disassemble Function



```
(gdb) disassemble function
Dump of assembler code for function function:
   0x56555590 <+0>:     push   %ebp
   0x56555591 <+1>:     mov    %esp,%ebp
   0x56555593 <+3>:     sub    $0x20,%esp
   0x56555596 <+6>:     call   0x5655560d <__x86.get_pc_thunk.ax>
   0x5655559b <+11>:    add    $0x1a65,%eax
   0x565555a0 <+16>:    movl   $0x1,-0x14(%ebp)
   0x565555a7 <+23>:    movl   $0x2,-0x10(%ebp)
   0x565555ae <+30>:    movl   $0x3,-0xc(%ebp)
   0x565555b5 <+37>:    movl   $0x4,-0x8(%ebp)
   0x565555bc <+44>:    movl   $0x5,-0x4(%ebp)
=> 0x565555c3 <+51>:    nop
   0x565555c4 <+52>:    leave
   0x565555c5 <+53>:    ret
End of assembler dump.
(gdb)
```

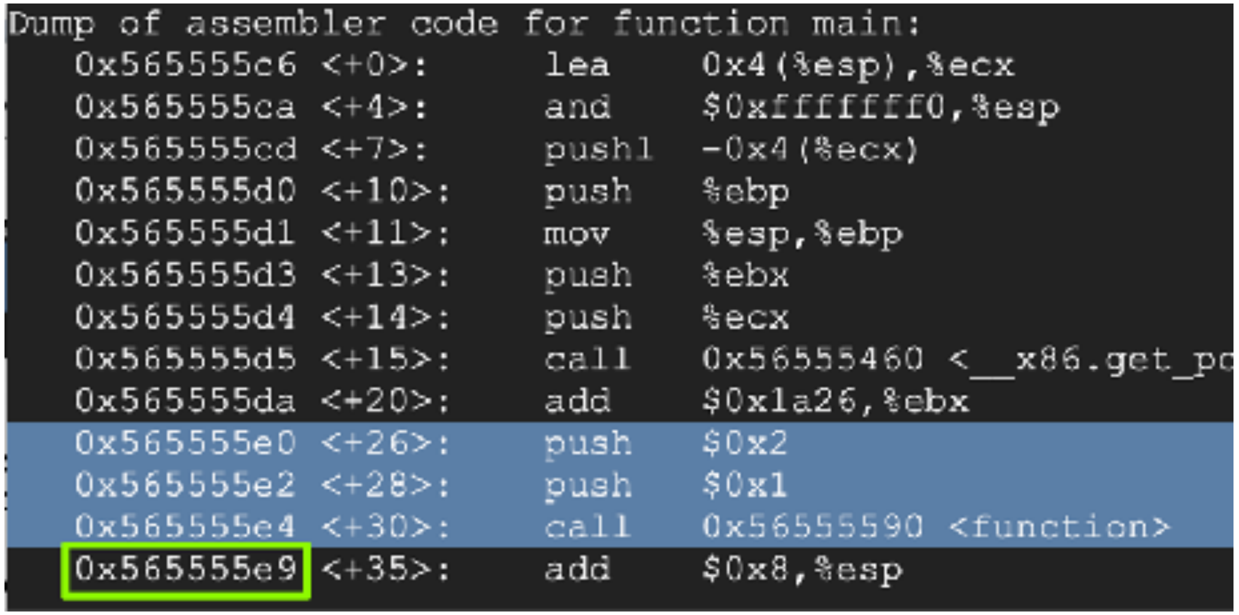
- Prolog:
  - **push** ebp onto stack
  - **mov** esp into ebp, starting a new stack frame
  - **sub** from esp, reserving room for local variables

Saved Return Address



```
(gdb) x/20x $esp
0xffffd620:     0xf7fc8000      0xffffd704      0xf7ffcd00      0x00000001
0xffffd630:     0x00000002      0x00000003      0x00000004      0x00000005
0xffffd640:     0xffffd658      0x565555e9      0x00000001      0x00000002
0xffffd650:     0xffffd670      0x00000000      0x00000000      0xf7e2c286
0xffffd660:     0x00000001      0xf7fc8000      0x00000000      0xf7e2c286
```

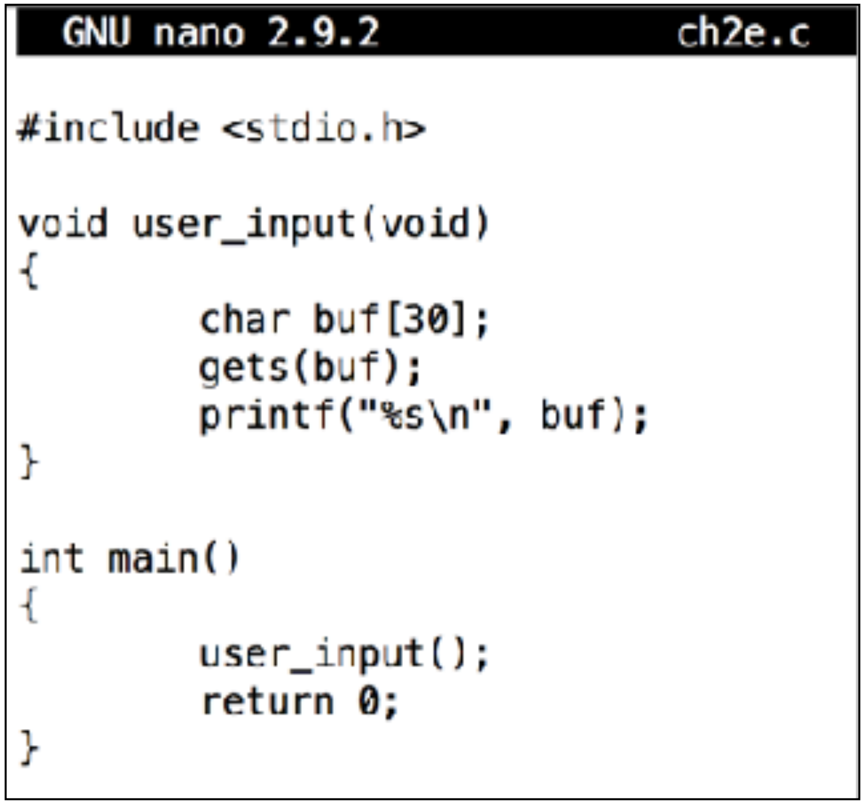
- Next word after stack frame
- Address of next instruction to be executed in main()



```
Dump of assembler code for function main:
   0x565555c6 <+0>:     lea    0x4(%esp),%ecx
   0x565555ca <+4>:     and    $0xfffffff0,%esp
   0x565555cd <+7>:     pushl  -0x4(%ecx)
   0x565555d0 <+10>:    push   %ebp
   0x565555d1 <+11>:    mov    %esp,%ebp
   0x565555d3 <+13>:    push   %ebx
   0x565555d4 <+14>:    push   %ecx
   0x565555d5 <+15>:    call   0x56555460 <__x86.get_pc
   0x565555da <+20>:    add    $0x1a26,%ebx
   0x565555e0 <+26>:    push   $0x2
   0x565555e2 <+28>:    push   $0x1
   0x565555e4 <+30>:    call   0x56555590 <function>
   0x565555e9 <+35>:    add    $0x8,%esp
```

Stack Buffer Overflow Exploit

Stack Buffer Overflow Vulnerability

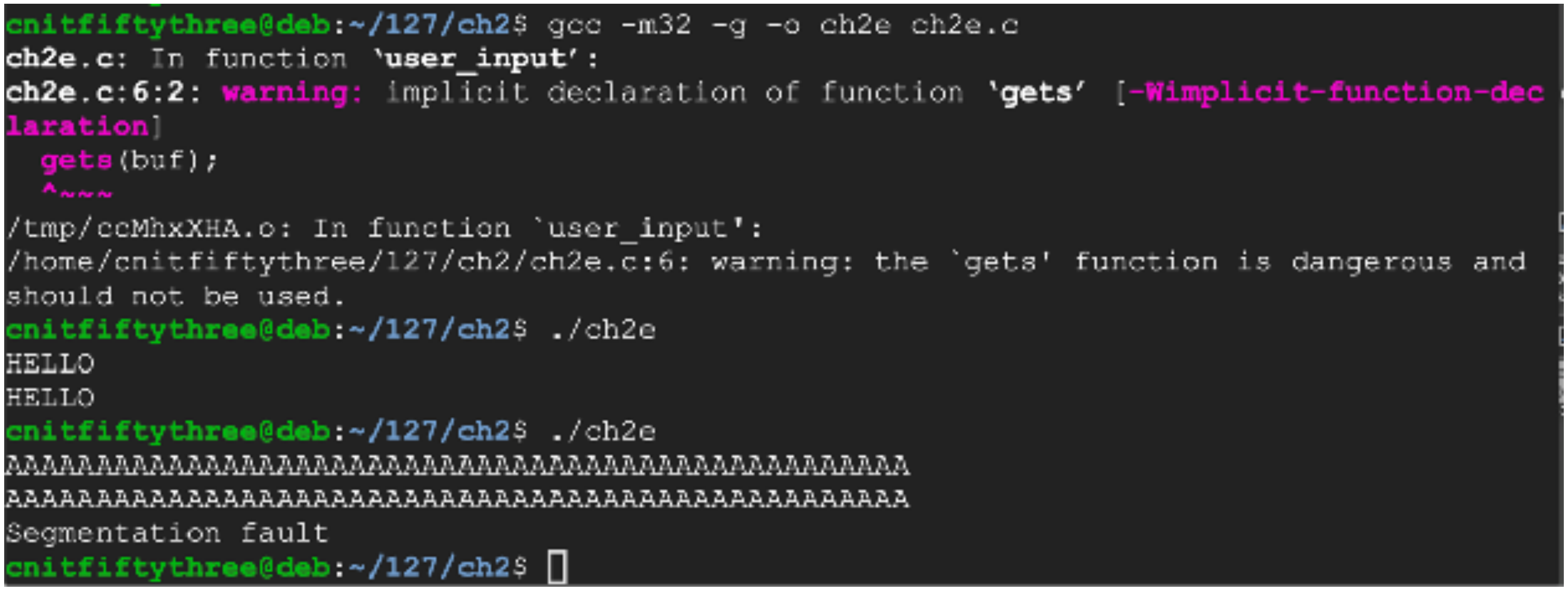


```
  GNU nano 2.9.2                ch2e.c

#include <stdio.h>

void user_input(void)
{
        char buf[30];
        gets(buf);
        printf("%s\n", buf);
}

int main()
{
        user_input();
        return 0;
}
```

gets() reads user input

Does not limit its length

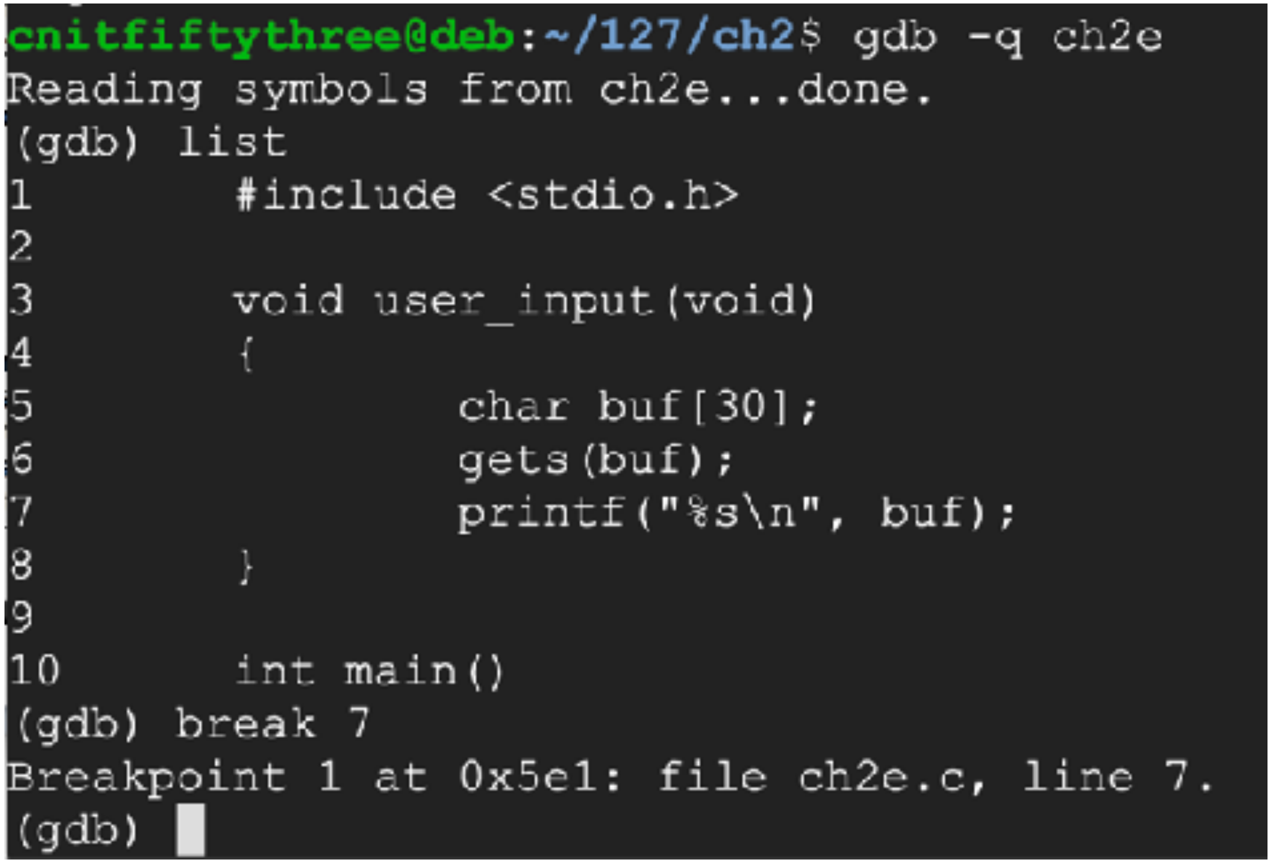
Compile and Run



```
cnitfiftythree@deb:~/127/ch2$ gcc -m32 -g -o ch2e ch2e.c
ch2e.c: In function 'user_input':
ch2e.c:6:2: warning: implicit declaration of function 'gets' [-Wimplicit-function-dec
laration]
  gets(buf);
  ^~~~
/tmp/ccMhxXHA.o: In function `user_input':
/home/cnitfiftythree/127/ch2/ch2e.c:6: warning: the `gets' function is dangerous and
should not be used.
cnitfiftythree@deb:~/127/ch2$ ./ch2e
HELLO
HELLO
cnitfiftythree@deb:~/127/ch2$ ./ch2e
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
cnitfiftythree@deb:~/127/ch2$ 
```

Segmentation fault indicates an illegal operation

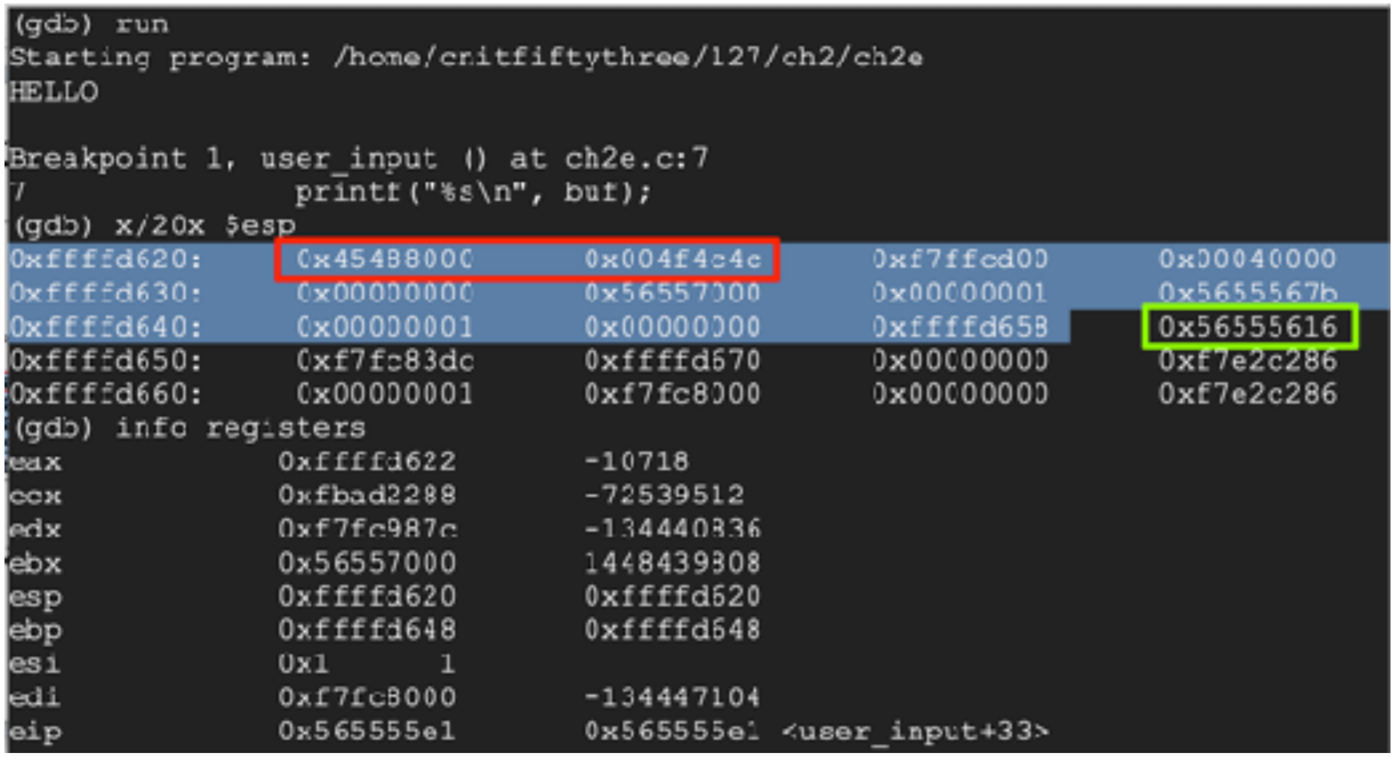
Debug and Set Breakpoint



```
cnitfiftythree@deb:~/127/ch2$ gdb -q ch2e
Reading symbols from ch2e...done.
(gdb) list
1       #include <stdio.h>
2
3       void user_input(void)
4       {
5               char buf[30];
6               gets(buf);
7               printf("%s\n", buf);
8       }
9
10      int main()
(gdb) break 7
Breakpoint 1 at 0x5e1: file ch2e.c, line 7.
(gdb) 
```

Break after **gets()**

Stack After **HELLO**

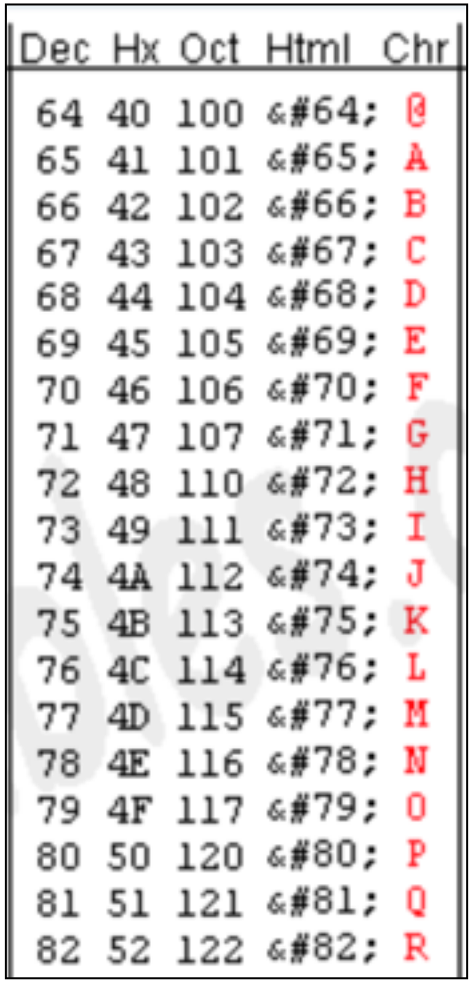


```
(gdb) run
Starting program: /home/critfiftythree/127/ch2/ch2e
HELLO

Breakpoint 1, user_input () at ch2e.c:7
7               printf("%s\n", buf);
(gdb) x/20x $esp
0xffffd620:     0x45488000      0x004f4c4c      0xf7ffcd00      0x00040000
0xffffd630:     0x00000000      0x56557000      0x00000001      0x5655567b
0xffffd640:     0x00000001      0x00000000      0xffffd658      0x56555616
0xffffd650:     0xf7fc83dc      0xffffd670      0x00000000      0xf7e2c286
0xffffd660:     0x00000001      0xf7fc8000      0x00000000      0xf7e2c286
(gdb) info registers
eax            0xffffd622       -10718
ecx            0xfbad2288       -72539512
edx            0xf7fc987c       -134440836
ebx            0x56557000       1448439808
esp            0xffffd620       0xffffd620
ebp            0xffffd648       0xffffd648
esi            0x1      1
edi            0xf7fc8000       -134447104
eip            0x565555e1       0x565555e1 <user_input+33>
```

- ASCII values for **HELLO** appear in the words outlined in red
- Return value is outlined in green

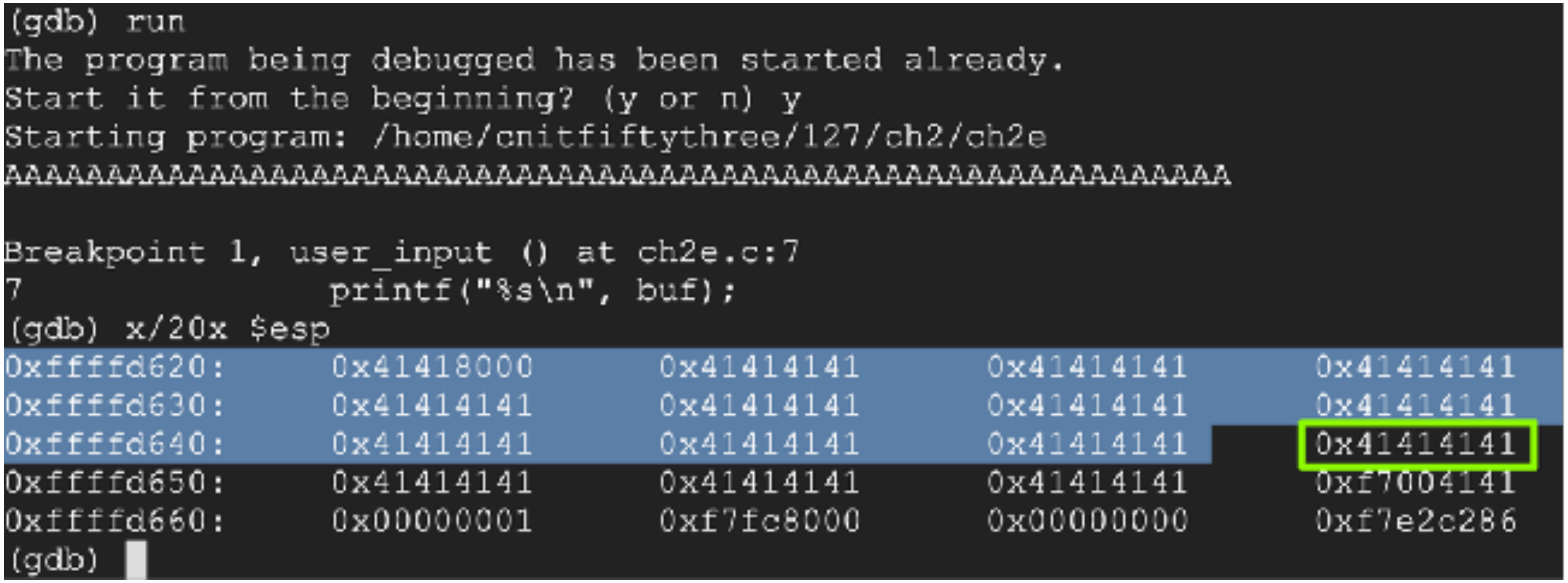
ASCII

- Google "ASCII"
  - 0x41 is **A**
  - 0x42 is **B**
  - etc.



Dec	Hx	Oct	Html	Chr
64	40	100	&#64;	@
65	41	101	&#65;	A
66	42	102	&#66;	B
67	43	103	&#67;	C
68	44	104	&#68;	D
69	45	105	&#69;	E
70	46	106	&#70;	F
71	47	107	&#71;	G
72	48	110	&#72;	H
73	49	111	&#73;	I
74	4A	112	&#74;	J
75	4B	113	&#75;	K
76	4C	114	&#76;	L
77	4D	115	&#77;	M
78	4E	116	&#78;	N
79	4F	117	&#79;	O
80	50	120	&#80;	P
81	51	121	&#81;	Q
82	52	122	&#82;	R

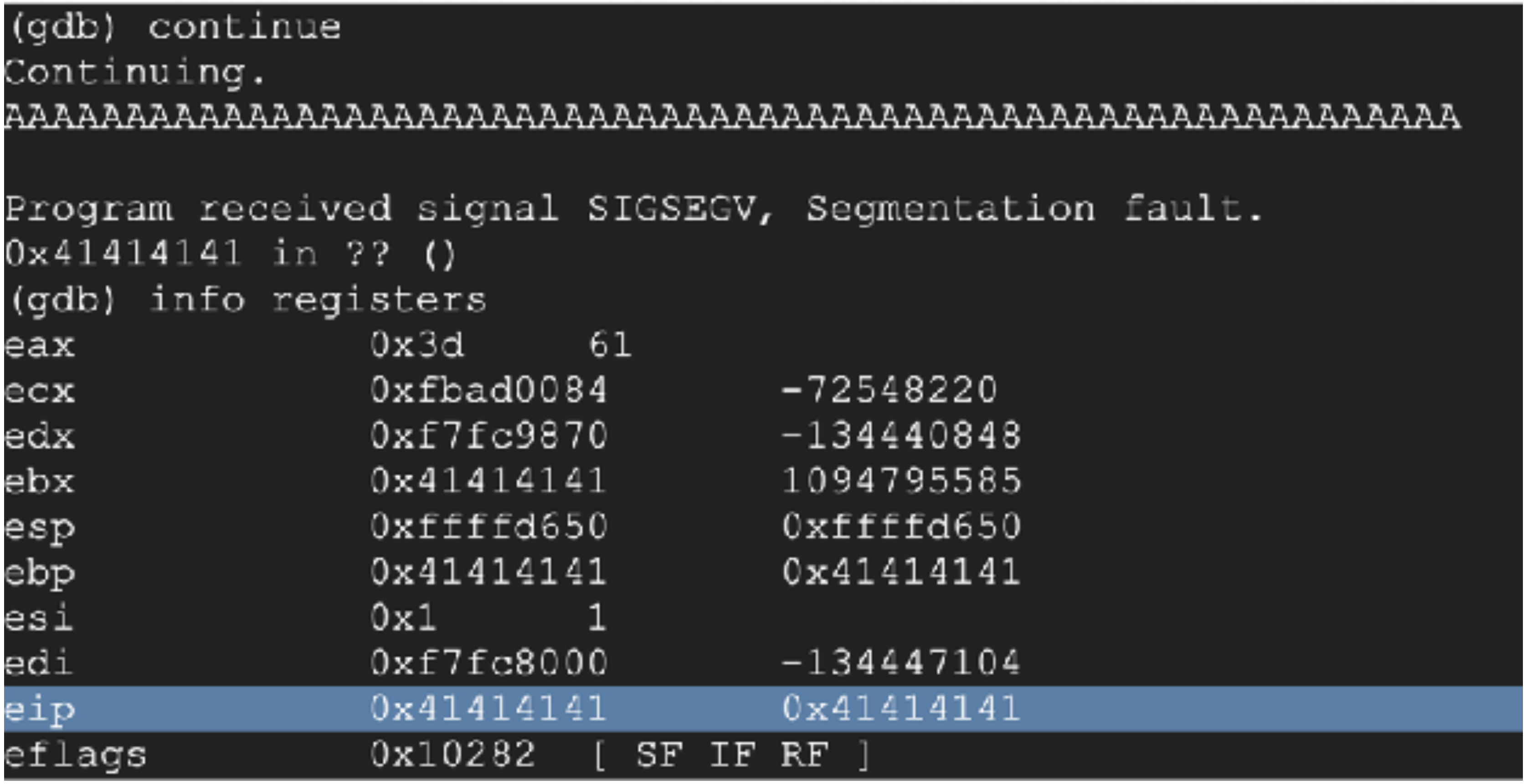
Stack After **AAAAA...**



```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/cnitfiftythree/127/ch2/ch2e
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, user_input () at ch2e.c:7
7               printf("%s\n", buf);
(gdb) x/20x $esp
0xffffd620:     0x41418000      0x41414141      0x41414141      0x41414141
0xffffd630:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd640:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd650:     0x41414141      0x41414141      0x41414141      0xf7004141
0xffffd660:     0x00000001      0xf7fc8000      0x00000000      0xf7e2c286
(gdb)
```

- Stack frame is filled with many A characters
- Return value is overwritten with **0x41414141**

Examining the Crash



```
(gdb) continue
Continuing.
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) info registers
eax            0x3d     61
ecx            0xfbad0084       -72548220
edx            0xf7fc9870       -134440848
ebx            0x41414141       1094795585
esp            0xffffd650       0xffffd650
ebp            0x41414141       0x41414141
esi            0x1      1
edi            0xf7fc8000       -134447104
eip            0x41414141       0x41414141
eflags         0x10282  [ SF IF RF ]
```

- eip value is **0x41414141**
- Controlled by user input!

gdb Commands

list	*show source code*
run	*execute program*
break	*insert breakpoint*
x	*examine memory*
disassemble	*show asssembly code*
continue	*resume execution*
info registers	*see registers*
info proc mapping	*see memory map*

Kahoot!