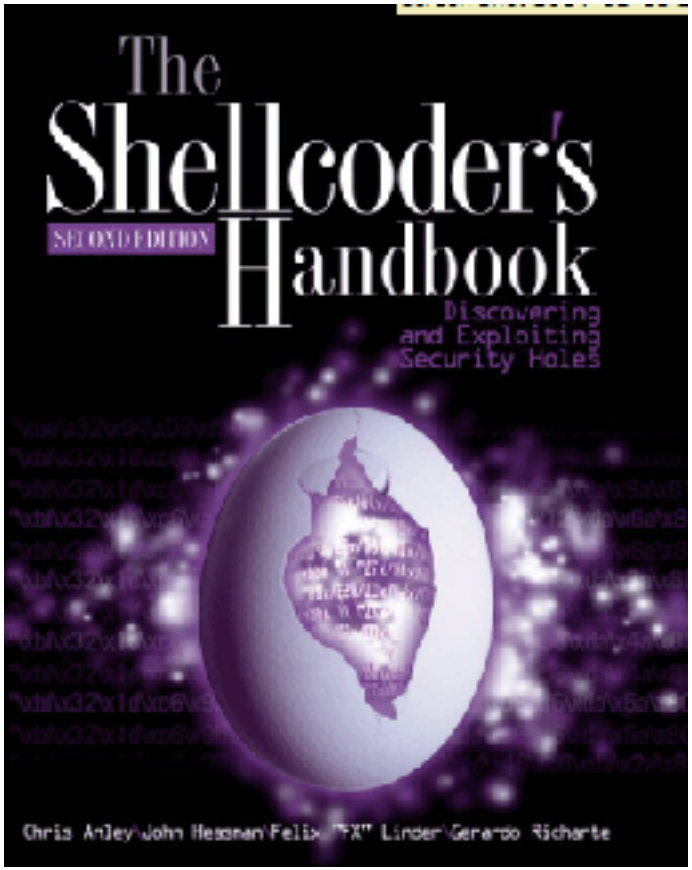
CNIT 127: Exploit Development

Ch 2: Stack Overflows in Linux



Updated 2-1-22

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*

Preparing a Debian Machine

- Tools needed to compile in 32-bit and debug

```
sudo apt update
sudo apt install build-essential gcc-multilib gdb -y
```

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

Reading Past End of Array

```
  GNU nano 3.2                                  ch2b.c

#include <stdio.h>

int main()
{
        int i, array[5] ;
        for (i=0; i<10000; i+=64)
                printf("%x %x\n", &array[i], array[i]);
}
```

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

Reading Past End of Array

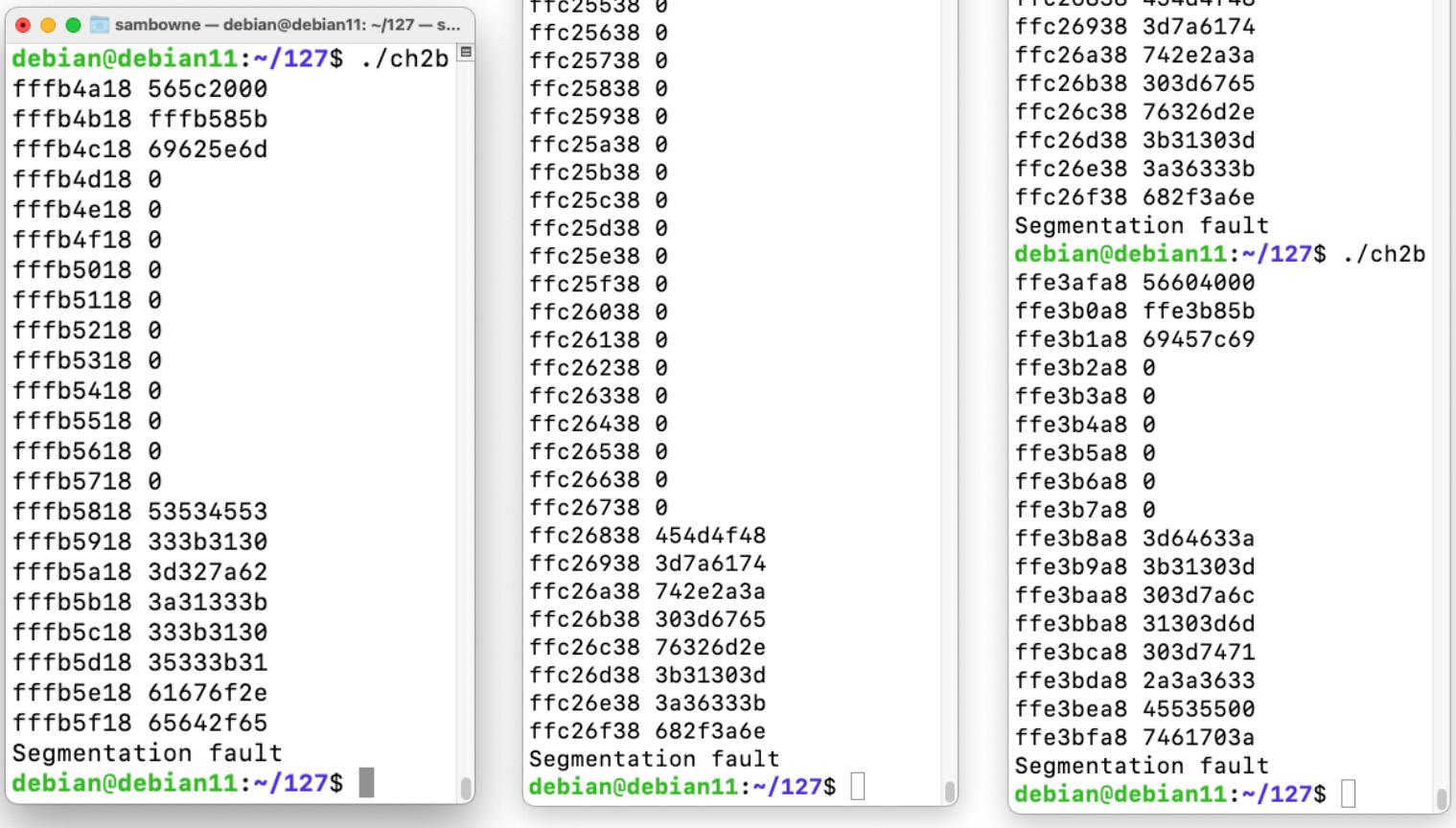
```
debian@debian10:~/127/ch2$ gcc -m32 -g -o ch2b ch2b.c
debian@debian10:~/127/ch2$ ./ch2b
ffff2808 ffff28dc
ffff2908 ffff4ed9
ffff2a08 0
ffff2b08 0
```

```
ffff4d08 62766d72
ffff4e08 333b3030
ffff4f08 72657375
Segmentation fault
```

- Program has crashed
- *Denial of service*

ASLR (Address Space Layout Randomization)

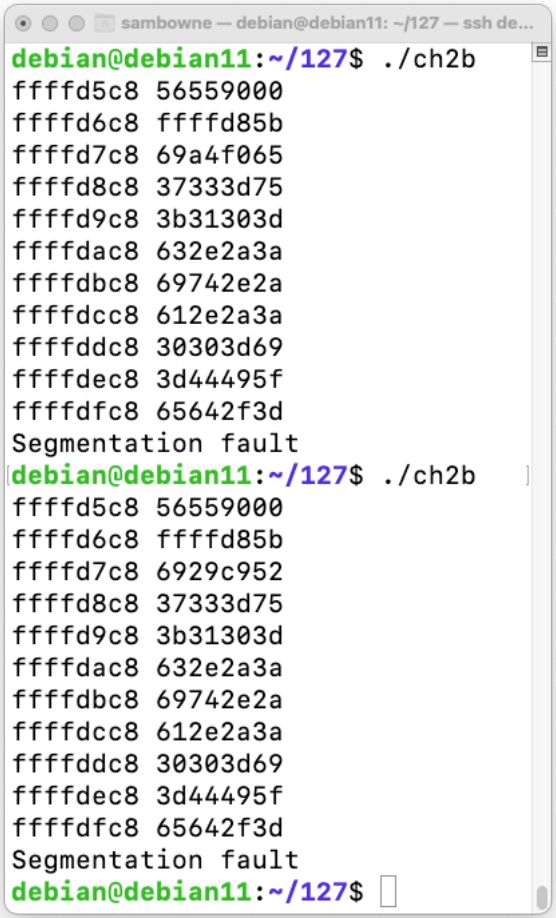
- Run ch2b several times
- The addresses change



Disabling ASLR

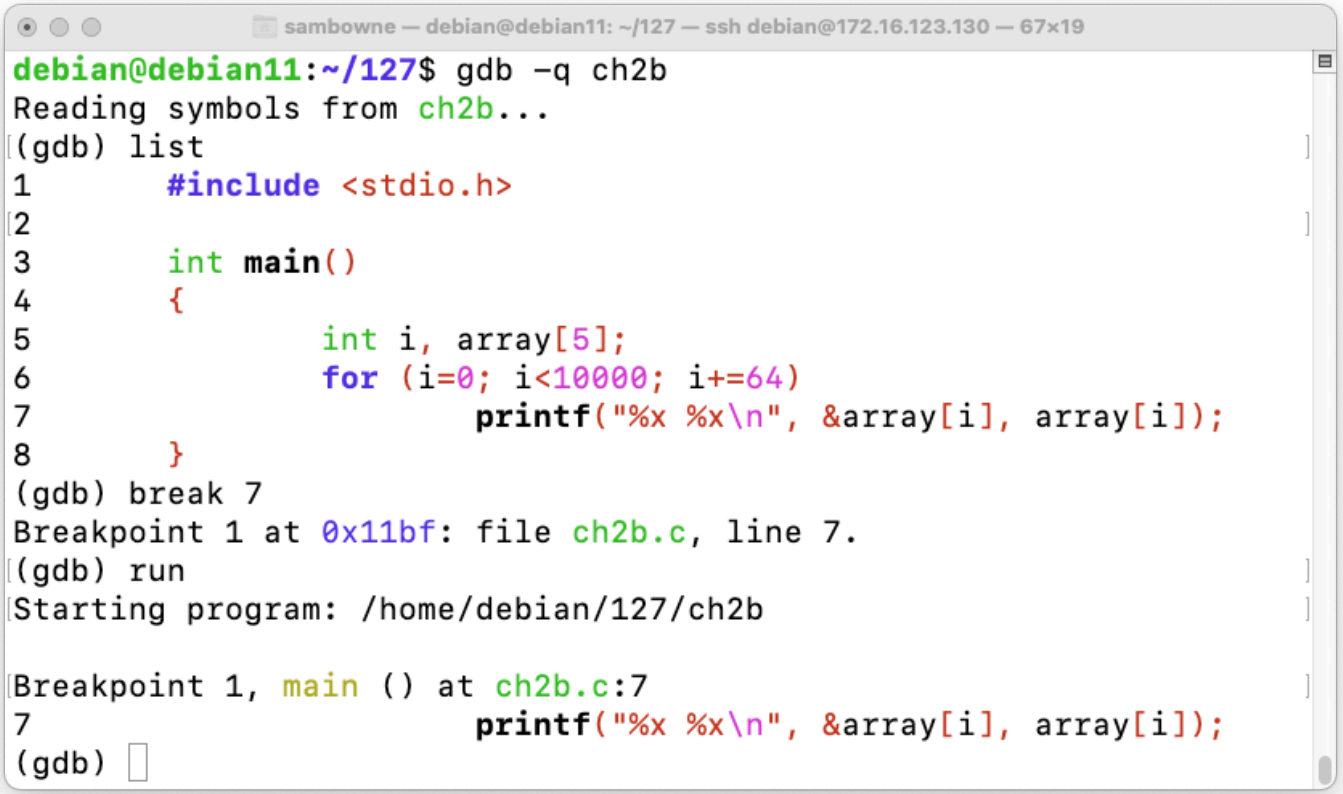
```
sudo su -
echo 0 > /proc/sys/kernel/randomize_va_space
exit
```

- Now the addresses are the same every time
- Crashes at ffffe000

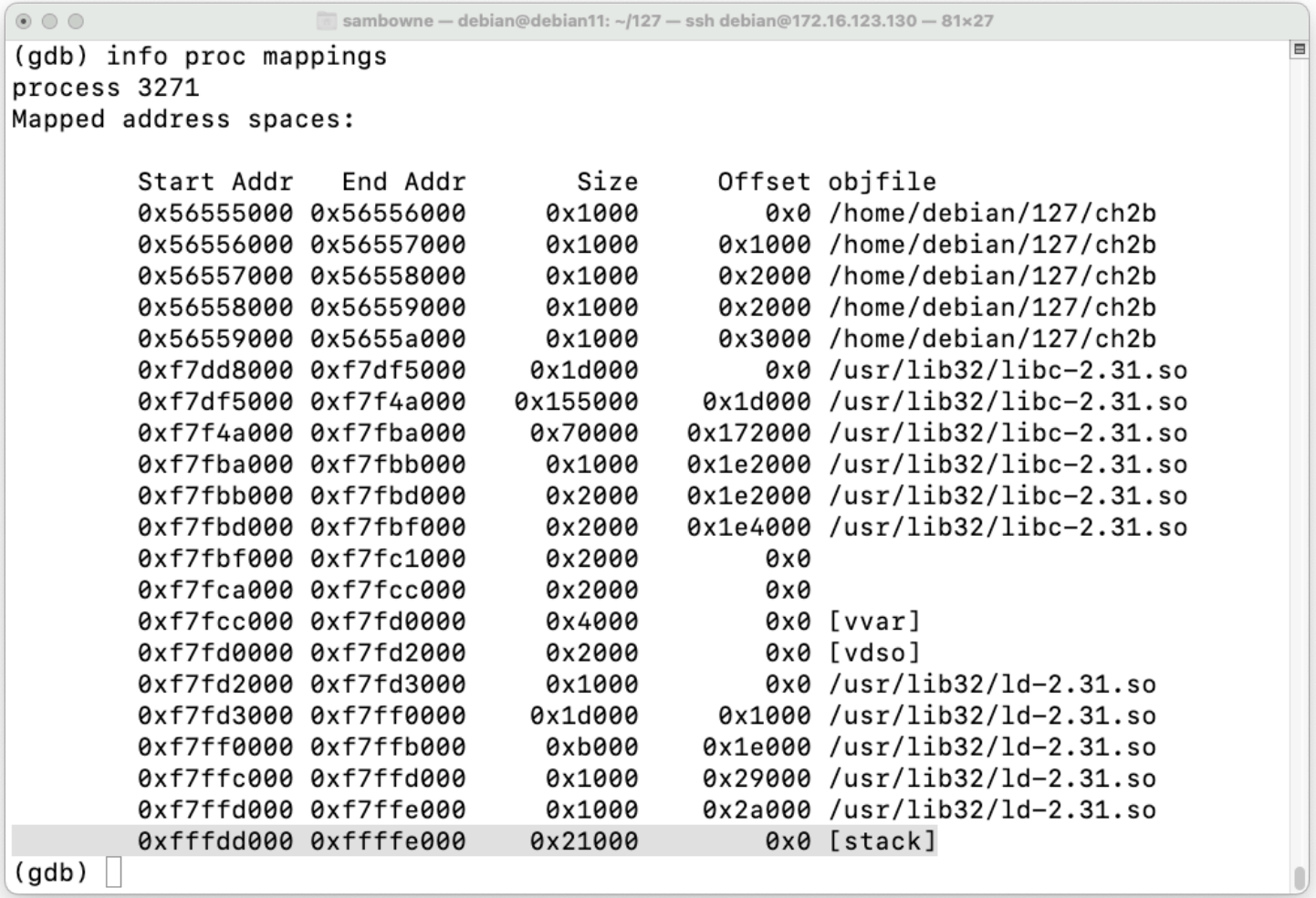


Debug

Insert breakpoint and run



Memory Map

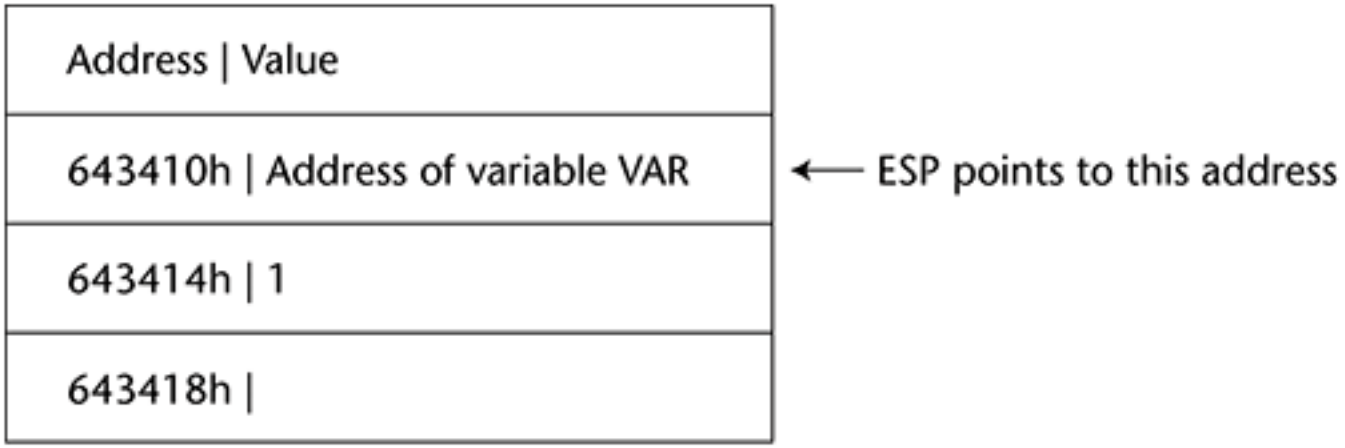


- **Stack** ends at **0xffffe000**
- Trying to read past this address caused a segmentation fault

The Stack

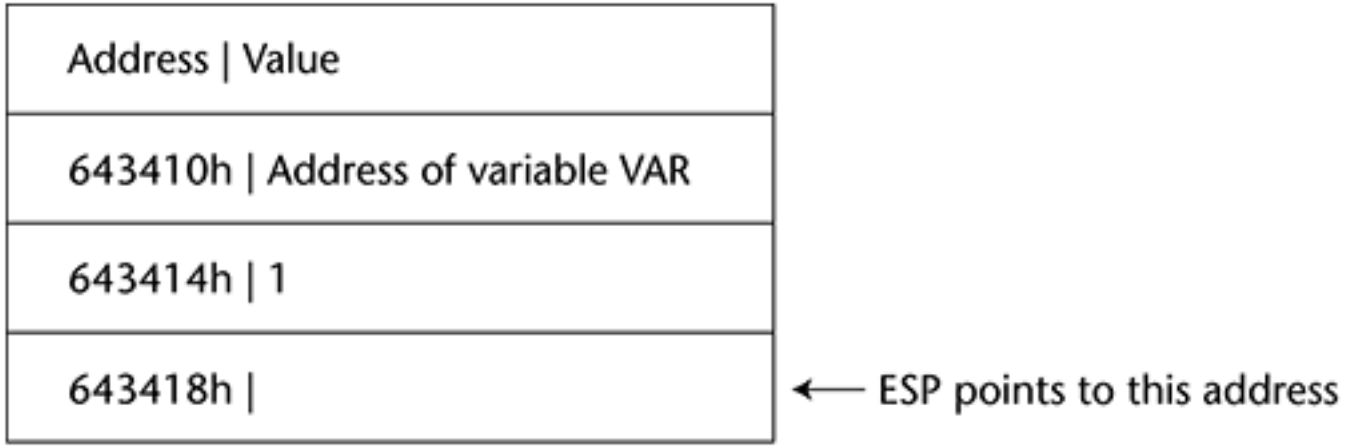
LIFO (Last-In, First-Out)

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



Stack

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



EBP (Extended Base Pointer)

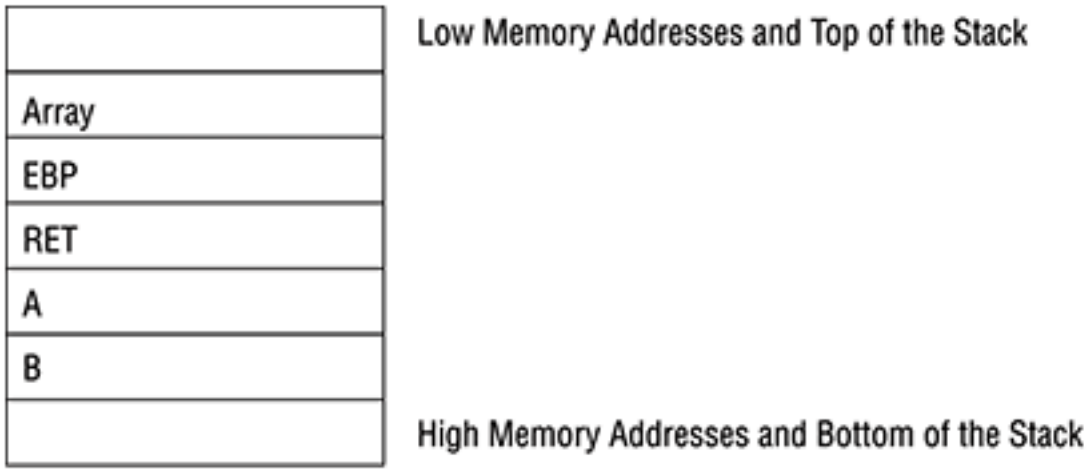
- EBP is typically used for calculated addresses on the stack
  - mov eax, [ebp+10h]
- Copies the data 16 bytes down the stack into the EAX register

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



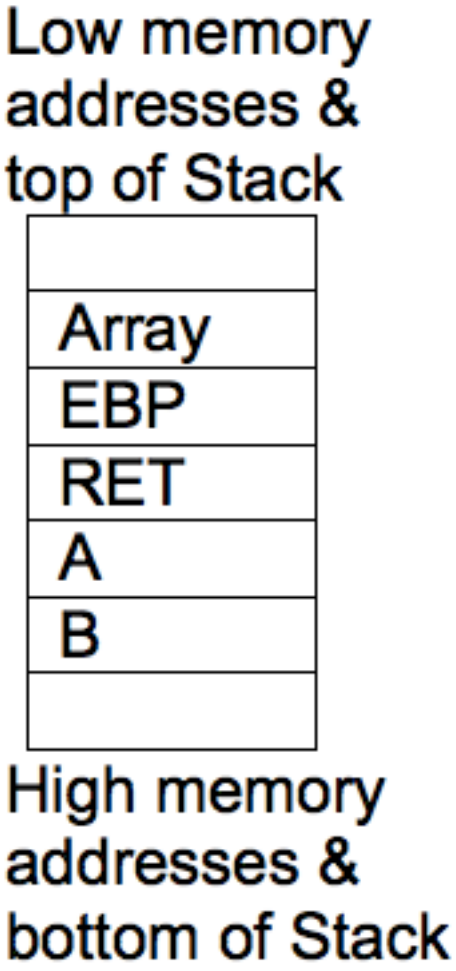
Functions and the Stack

- 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




```
#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 3.2                                  ch2c.c

#include <stdio.h>

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

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

```
debian@debian10:~/127/ch2$ gcc -m32 -g -o ch2c ch2c.c
debian@debian10:~/127/ch2$ ./ch2c
Returned from function
```

Debug and Set Breakpoints

```
debian@debian10:~/127/ch2$ gdb -q ch2c
Reading symbols from ch2c...done.
(gdb) list 1,12
1       #include <stdio.h>
2
3       void f(int a, int b)
4       {
5               int array[5] = {1, 2, 3, 4, 5};
6       }
7
8       int main()
9       {
10              f(1, 2);
11              printf("Returned from function\n");
12      }
(gdb) break 10
Breakpoint 1 at 0x11e9: file ch2c.c, line 10.
(gdb) break 6
Breakpoint 2 at 0x11cc: file ch2c.c, line 6.
(gdb)
```

In main()

```
(gdb) run
Starting program: /home/debian/127/ch2/ch2c

Breakpoint 1, main () at ch2c.c:10
10              f(1, 2);
(gdb) info registers
eax            0xf7fb1dc8          -134537784
ecx            0xffffd610          -10736
edx            0xffffd634          -10700
ebx            0x56559000          1448448000
esp            0xffffd5f0          0xffffd5f0
ebp            0xffffd5f8          0xffffd5f8
esi            0xf7fb0000          -134545408
edi            0xf7fb0000          -134545408
eip            0x565561e9          0x565561e9 <main+26>
```

Stack frame goes from esp to ebp

In function()

```
(gdb) continue
Continuing.

Breakpoint 2, f (a=1, b=2) at ch2c.c:6
6       }
(gdb) info registers
eax            0x56559000          1448448000
ecx            0xffffd610          -10736
edx            0xffffd634          -10700
ebx            0x56559000          1448448000
esp            0xffffd5c0          0xffffd5c0
ebp            0xffffd5e0          0xffffd5e0
esi            0xf7fb0000          -134545408
edi            0xf7fb0000          -134545408
eip            0x565561cc          0x565561cc <f+51>
```

Stack frame goes from esp to ebp

Examine the Stack Frame

```
(gdb) info registers
eax            0x56559000          1448448000
ecx            0xffffd610          -10736
edx            0xffffd634          -10700
ebx            0x56559000          1448448000
esp            0xffffd5c0          0xffffd5c0
ebp            0xffffd5e0          0xffffd5e0
esi            0xf7fb0000          -134545408
edi            0xf7fb0000          -134545408
eip            0x565561cc          0x565561cc <f+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
0xffffd5c0:     0xf7fb0000      0xf7fb0000      0x00000000      0x00000001
0xffffd5d0:     0x00000002      0x00000003      0x00000004      0x00000005
0xffffd5e0:     0xffffd5f8      0x565561f2      0x00000001      0x00000002
0xffffd5f0:     0xffffd610      0x00000000      0x00000000      0xf7df0b41
0xffffd600:     0xf7fb0000      0xf7fb0000      0x00000000      0xf7df0b41
(gdb)
```

- Highlighted region is the **stack frame** of function()
- The next word is the **return pointer**

Disassemble Main

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x565561cf <+0>:     lea    0x4(%esp),%ecx
   0x565561d3 <+4>:     and    $0xfffffff0,%esp
   0x565561d6 <+7>:     pushl  -0x4(%ecx)
   0x565561d9 <+10>:    push   %ebp
   0x565561da <+11>:    mov    %esp,%ebp
   0x565561dc <+13>:    push   %ebx
   0x565561dd <+14>:    push   %ecx
   0x565561de <+15>:    call   0x565560a0 <__x86.get_pc_thunk.bx>
   0x565561e3 <+20>:    add    $0x2e1d,%ebx
   0x565561e9 <+26>:    push   $0x2
   0x565561eb <+28>:    push   $0x1
   0x565561ed <+30>:    call   0x56556199 <f>
   0x565561f2 <+35>:    add    $0x8,%esp
```

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

Disassemble Function

```
(gdb) disassemble f
Dump of assembler code for function f:
   0x56556199 <+0>:     push   %ebp
   0x5655619a <+1>:     mov    %esp,%ebp
   0x5655619c <+3>:     sub    $0x20,%esp
   0x5655619f <+6>:     call   0x56556216 <__x86.get_pc_thunk.ax>
   0x565561a4 <+11>:    add    $0x2e5c,%eax
   0x565561a9 <+16>:    movl   $0x1,-0x14(%ebp)
   0x565561b0 <+23>:    movl   $0x2,-0x10(%ebp)
   0x565561b7 <+30>:    movl   $0x3,-0xc(%ebp)
   0x565561be <+37>:    movl   $0x4,-0x8(%ebp)
   0x565561c5 <+44>:    movl   $0x5,-0x4(%ebp)
=> 0x565561cc <+51>:    nop
   0x565561cd <+52>:    leave
   0x565561ce <+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
0xffffd5c0:     0xf7fb0000      0xf7fb0000      0x00000000      0x00000001
0xffffd5d0:     0x00000002      0x00000003      0x00000004      0x00000005
0xffffd5e0:     0xffffd5f8      0x565561f2      0x00000001      0x00000002
0xffffd5f0:     0xffffd610      0x00000000      0x00000000      0xf7df0b41
0xffffd600:     0xf7fb0000      0xf7fb0000      0x00000000      0xf7df0b41
(gdb)
```

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

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x565561cf <+0>:     lea    0x4(%esp),%ecx
   0x565561d3 <+4>:     and    $0xfffffff0,%esp
   0x565561d6 <+7>:     pushl  -0x4(%ecx)
   0x565561d9 <+10>:    push   %ebp
   0x565561da <+11>:    mov    %esp,%ebp
   0x565561dc <+13>:    push   %ebx
   0x565561dd <+14>:    push   %ecx
   0x565561de <+15>:    call   0x565560a0 <__x86.get_pc_thunk.bx>
   0x565561e3 <+20>:    add    $0x2e1d,%ebx
   0x565561e9 <+26>:    push   $0x2
   0x565561eb <+28>:    push   $0x1
   0x565561ed <+30>:    call   0x56556199 <f>
   0x565561f2 <+35>:    add    $0x8,%esp
```

Stack Buffer Overflow Exploit

Stack Buffer Overflow Vulnerability

GNU nano 3.2 ch2d.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

```
debian@debian10:~/127/ch2$ gcc -m32 -g -o ch2d ch2d.c
ch2d.c: In function ‘user_input’:
ch2d.c:6:2: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wim
plicit-function-declaration]
  gets(buf);
  ^~~~
  fgets
/usr/bin/ld: /tmp/ccVA4e8C.o: in function `user_input':
/home/debian/127/ch2/ch2d.c:6: warning: the `gets' function is dangerous and should not b
e used.
debian@debian10:~/127/ch2$ ./ch2d
HELLO
HELLO
debian@debian10:~/127/ch2$ ./ch2d
AAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE
AAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE
Segmentation fault
```

Segmentation fault indicates an illegal operation

Debug and Set Breakpoint

```
debian@debian10:~/127/ch2$ gdb -q ch2d
Reading symbols from ch2d...done.
(gdb) list 1,14
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()
11      {
12              user_input();
13              return 0;
14      }
(gdb) break 7
Breakpoint 1 at 0x11ca: file ch2d.c, line 7.
(gdb)
```

Break after **gets()**

Stack After **HELLO**

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

Breakpoint 1, user_input () at ch2d.c:7
7               printf("%s\n", buf);
(gdb) x/20x $esp
0xffffd5d0:     0x454803fc      0x004f4c4c      0xffffd6ac      0x56556243
0xffffd5e0:     0x00000001      0xffffd6a4      0xffffd6ac      0x5655621b
0xffffd5f0:     0xf7fe4520      0x00000000      0xffffd608      0x565561f4
0xffffd600:     0xf7fb0000      0xf7fb0000      0x00000000      0xf7df0b41
0xffffd610:     0x00000001      0xffffd6a4      0xffffd6ac      0xffffd634
(gdb) info registers
eax            0xffffd5d2          -10798
ecx            0xf7fb05c0          -134543936
edx            0xf7fb189c          -134539108
ebx            0x56559000          1448448000
esp            0xffffd5d0          0xffffd5d0
ebp            0xffffd5f8          0xffffd5f8
```

- 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	@	@
65	41	101	A	A
66	42	102	B	B
67	43	103	C	C
68	44	104	D	D
69	45	105	E	E
70	46	106	F	F
71	47	107	G	G
72	48	110	H	H
73	49	111	I	I
74	4A	112	J	J
75	4B	113	K	K
76	4C	114	L	L
77	4D	115	M	M
78	4E	116	N	N
79	4F	117	O	O
80	50	120	P	P
81	51	121	Q	Q
82	52	122	R	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/debian/127/ch2/ch2d
AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE

Breakpoint 1, user_input () at ch2d.c:7
7               printf("%s\n", buf);
(gdb) x/20x $esp
0xffffd5d0:     0x414103fc      0x41414141      0x41414141      0x42424242
0xffffd5e0:     0x42424242      0x43434242      0x43434343      0x43434343
0xffffd5f0:     0x44444444      0x44444444      0x45454444      0x45454545
0xffffd600:     0x45454545      0xf7fb0000      0x00000000      0xf7df0b41
0xffffd610:     0x00000001      0xffffd6a4      0xffffd6ac      0xffffd634
(gdb)
```

- Stack frame is filled with letters
- Return value is overwritten with **0x45454545**

Examining the Crash

```
(gdb) continue
Continuing.
AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE

Program received signal SIGSEGV, Segmentation fault.
0x45454545 in ?? ()
(gdb) info registers
eax            0x33                51
ecx            0xf7fb1890          -134539120
edx            0x33                51
ebx            0x44444444          1145324612
esp            0xffffd600          0xffffd600
ebp            0x45454444          0x45454444
esi            0xf7fb0000          -134545408
edi            0xf7fb0000          -134545408
eip            0x45454545          0x45454545
```

- eip value is **0x45454545**
- 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*

CNIT 127 Ch 2