Inside Buffer Overflows

James D. Murray, CISSP C|EH

@jdmurray| www.TechExams.net/blogs/jdmurray

pluralsight

Inside Computer Programs

1000101011

42,657,212,33

password123

Data

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello %s!\n", argv[0]);
    return 0;
}
```

Code

pluralsight

Inside Computer Programs

1000101011

42,657,212,33

password123

Program Memory

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello %s!\n", argv[0]);
    return 0;
}
```

pluralsight

From Disk to RAM to CPU

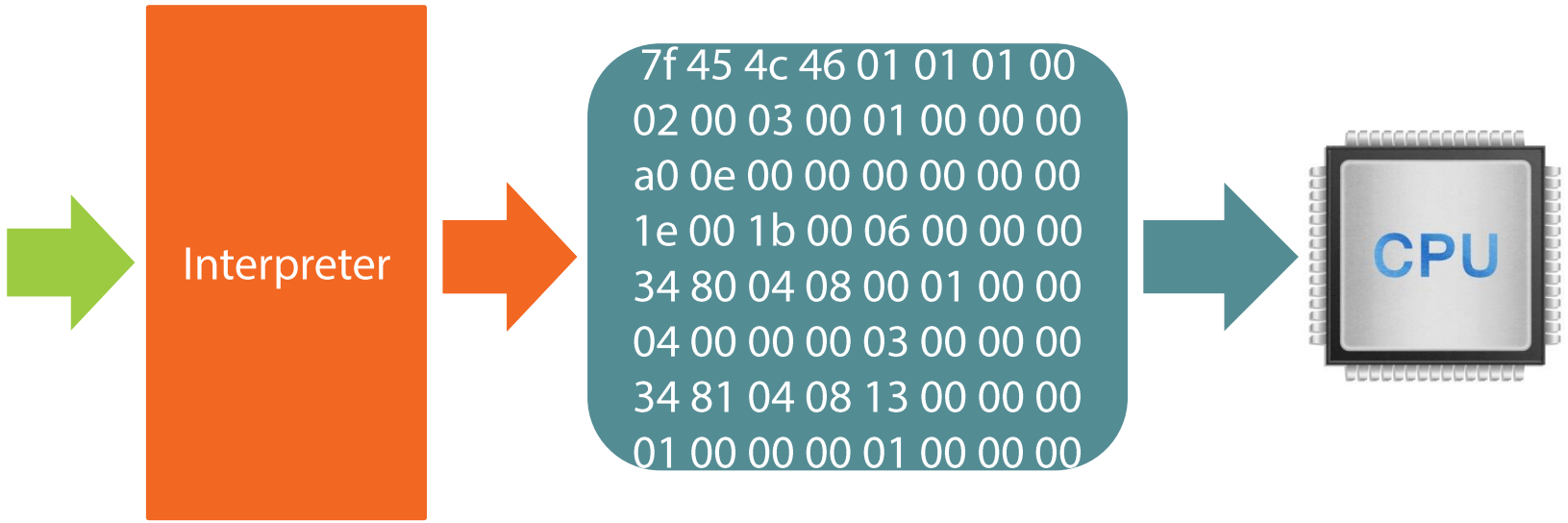
```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello %s!\n", argv[0]);
    return 0;
}
```

```
        .file   "hello.c"
        .section        .rodata
        .string "Hello %s!\n"
        .text
        .globl  main
        .type   main, @function
main:
        .cfi_startproc
```

pluralsight

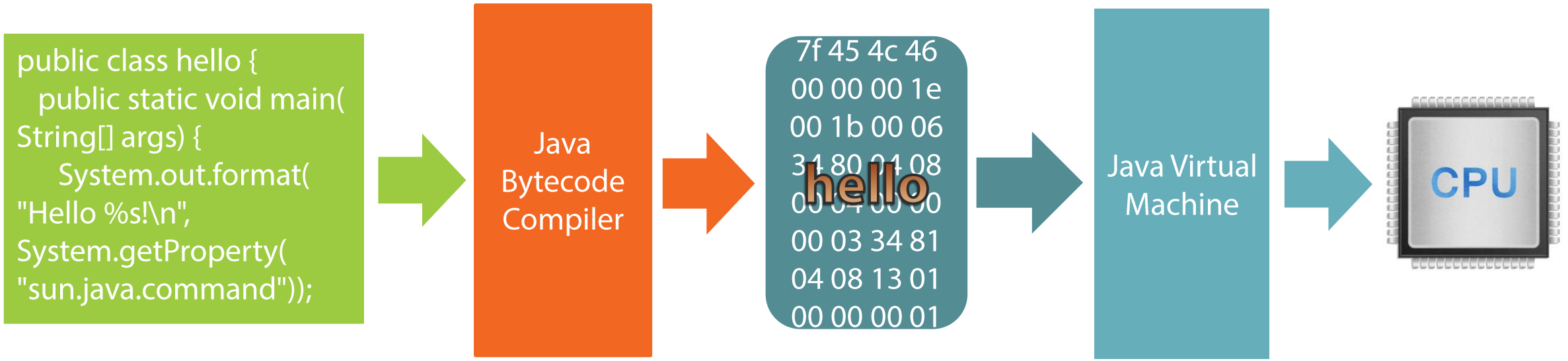
From Disk to RAM to CPU

```
#!/usr/bin/python
import sys
def main(argv):
        print 'Hello {0!s}!'.format(argv)
if __name__ == "__main__":
        main(sys.argv[0:])
```



pluralsight

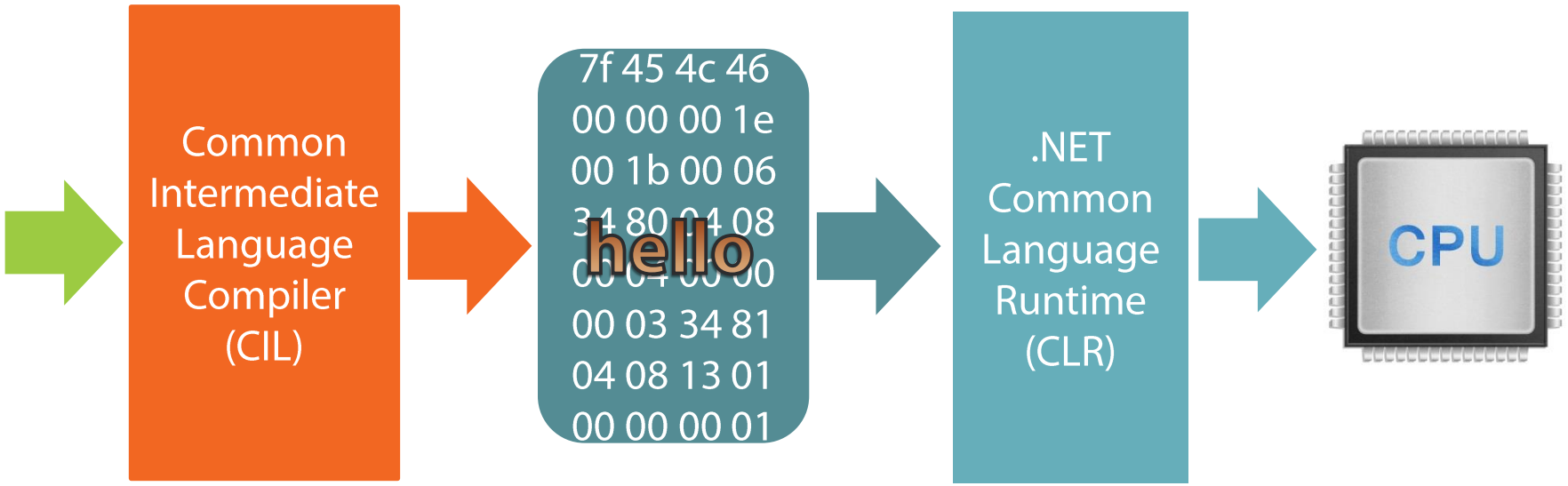
From Disk to RAM to CPU



pluralsight

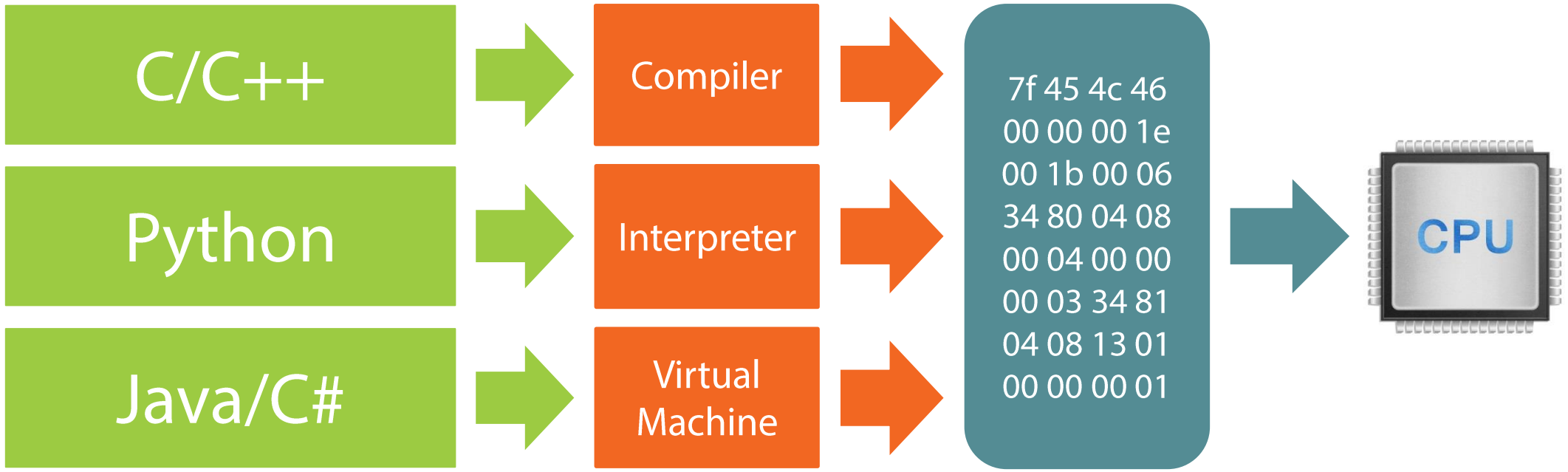
From Disk to RAM to CPU

```
using System;
public class hello
{
   public static int
Main(string[] args)
   {
    Console.WriteLine(
"Hello {0}!",
System.AppDomain.Current
Domain.FriendlyName);
      return 0;
   }
}
```



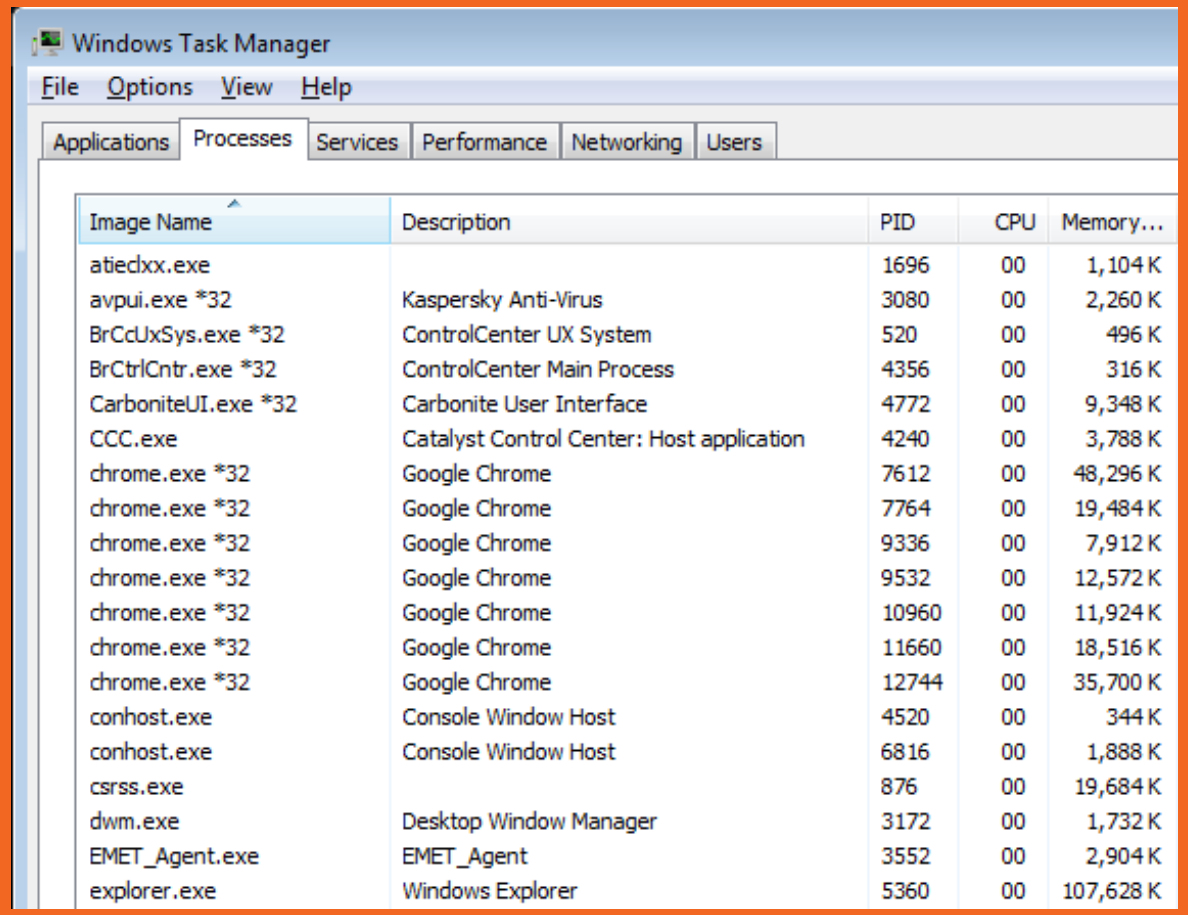
pluralsight

From Program to Process



pluralsight

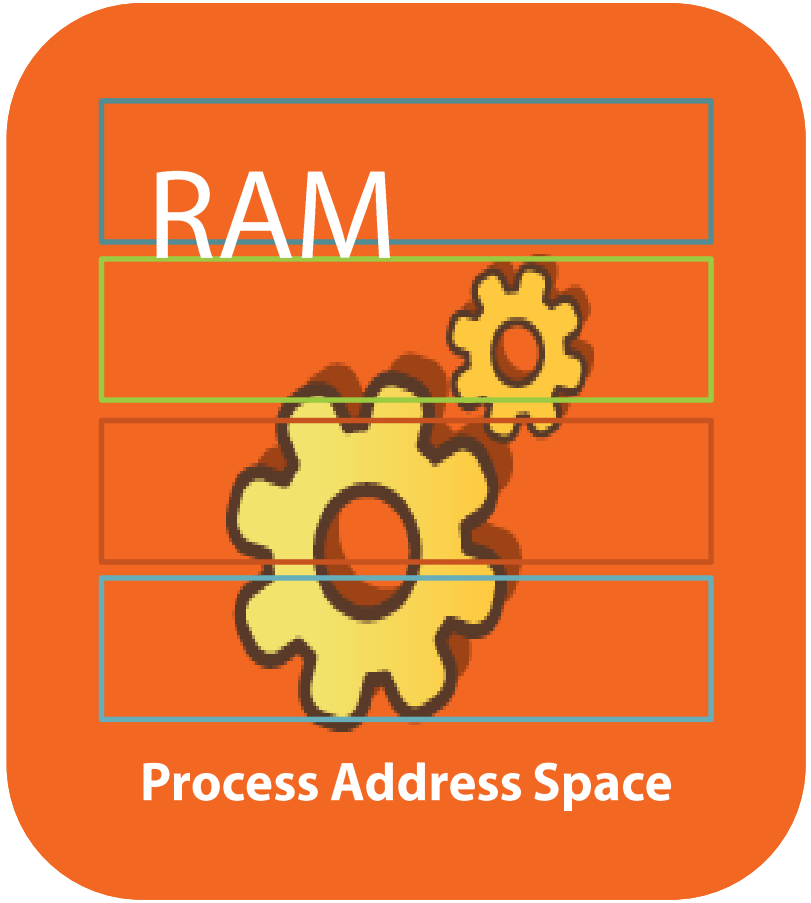
From Program to Process



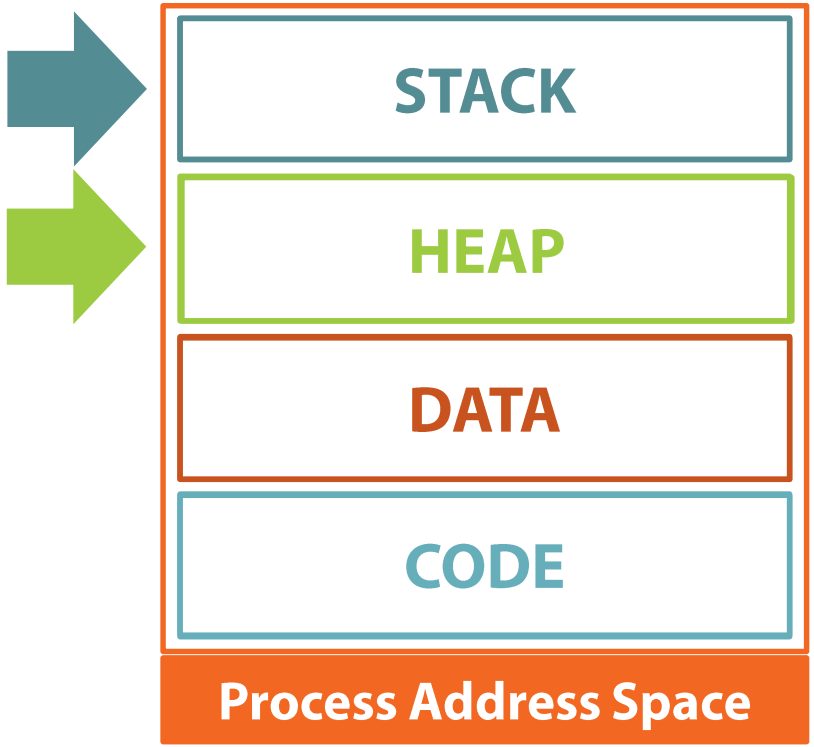

```
root@kali:~# ps -eH
  PID TTY          TIME CMD
    2 ?        00:00:00 kthreadd
    3 ?        00:00:23   ksoftirqd/0
    5 ?        00:00:00   kworker/0:0H
    7 ?        00:01:40   rcu_sched
    8 ?        00:00:00   rcu_bh
    9 ?        00:00:00   migration/0
   10 ?        00:00:01   watchdog/0
   11 ?        00:00:00   khelper
   12 ?        00:00:00   kdevtmpfs
   13 ?        00:00:00   netns
   14 ?        00:00:00   perf
   15 ?        00:00:00   khungtaskd
   16 ?        00:00:00   writeback
   17 ?        00:00:00   ksmd
   19 ?        00:00:00   khugepaged
   20 ?        00:00:00   crypto
```

pluralsight

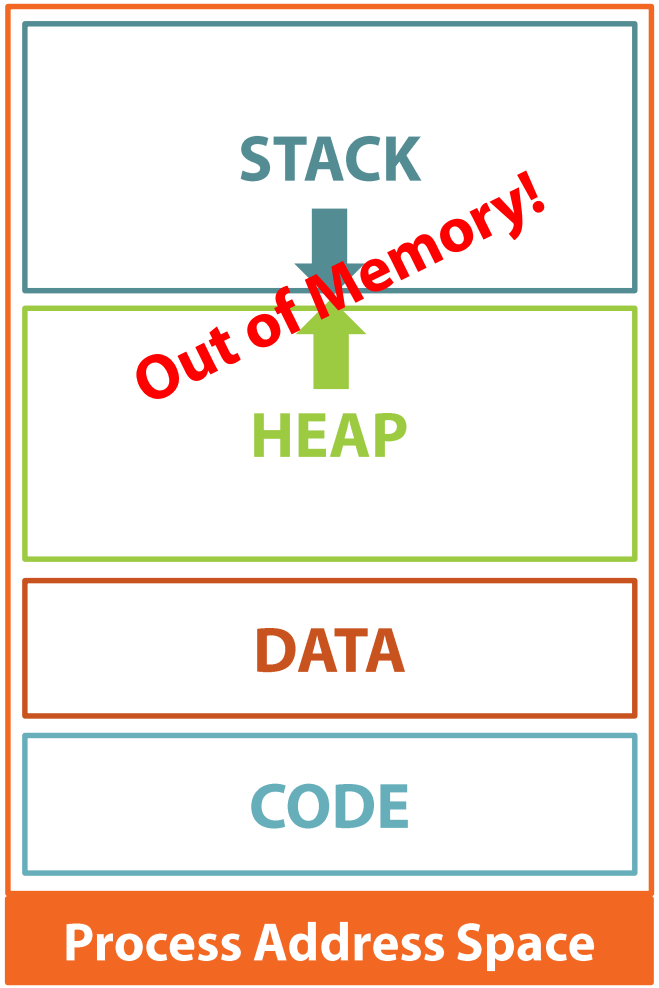
Inside Process Memory



Inside Process Memory



Fixed and Dynamic Memory



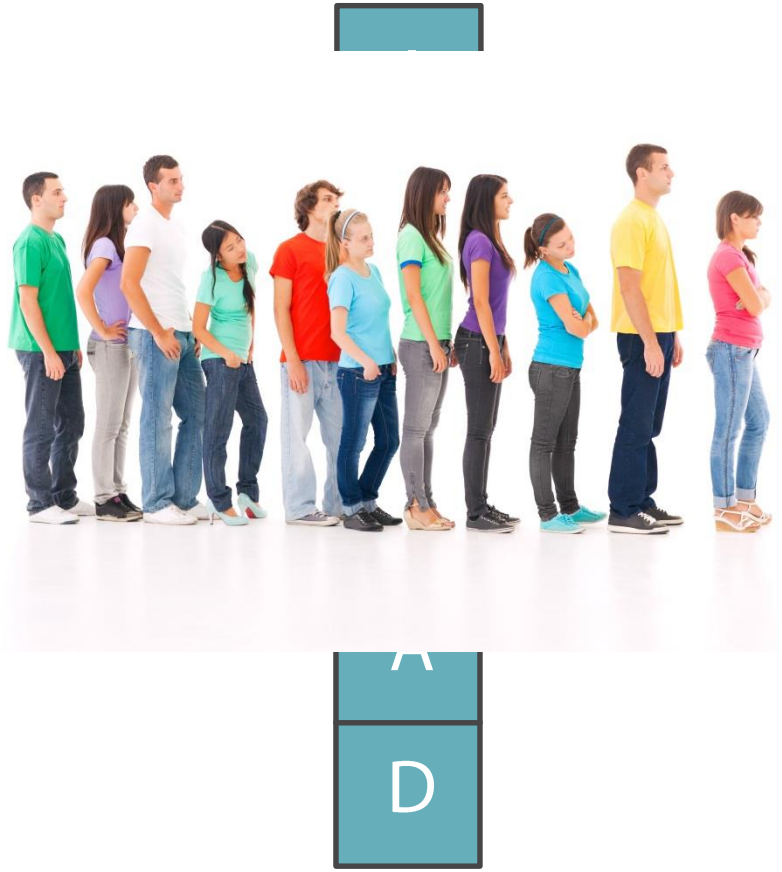
Inside the Stack

- Overflows are typically in the stack
- Stack overflow causes:
  - Process crash
  - Data leakage/exfiltration
  - Arbitrary code execution

pluralsight

Just Waiting in Another Queue

- A stack is a memory buffer
- Temporarily stores data
- Data may be in use now or used later
- Queue data structure
- Both people and data wait in queues



pluralsight

LIFO or FILO--It's Your Choice

LIFO

Last In First Out

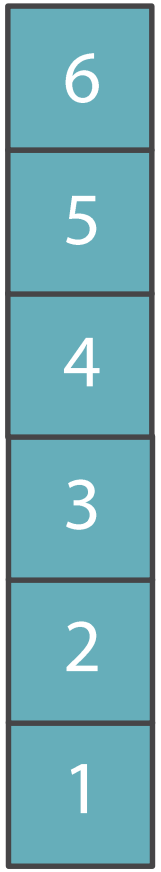
FILO

First In Last Out

pluralsight

Value and Order Make Unique

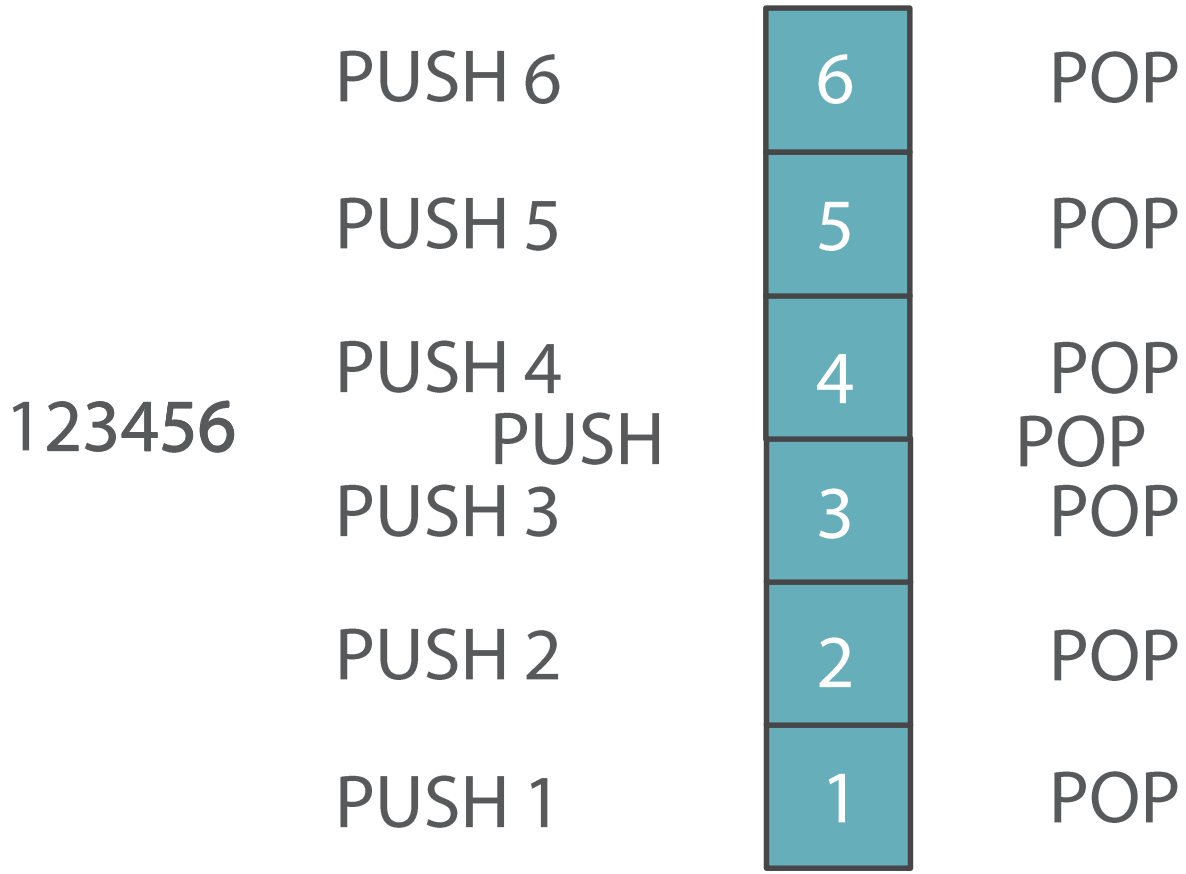
1 2 3 4 5 6



1 2 3 4 5 6

pluralsight

PUSH and POP



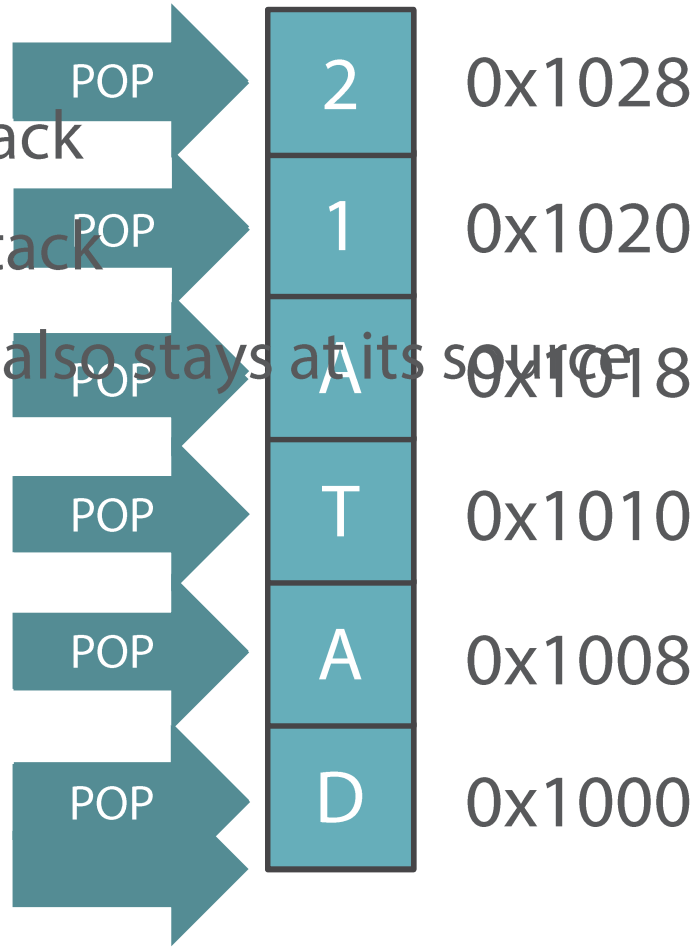
pluralsight

The Stack Pointer

- Push and Pop move data?
- Push copies data to the stack
- Pop reads data from the stack
- Data pushed and popped also stays at its source

SP = 0x1028

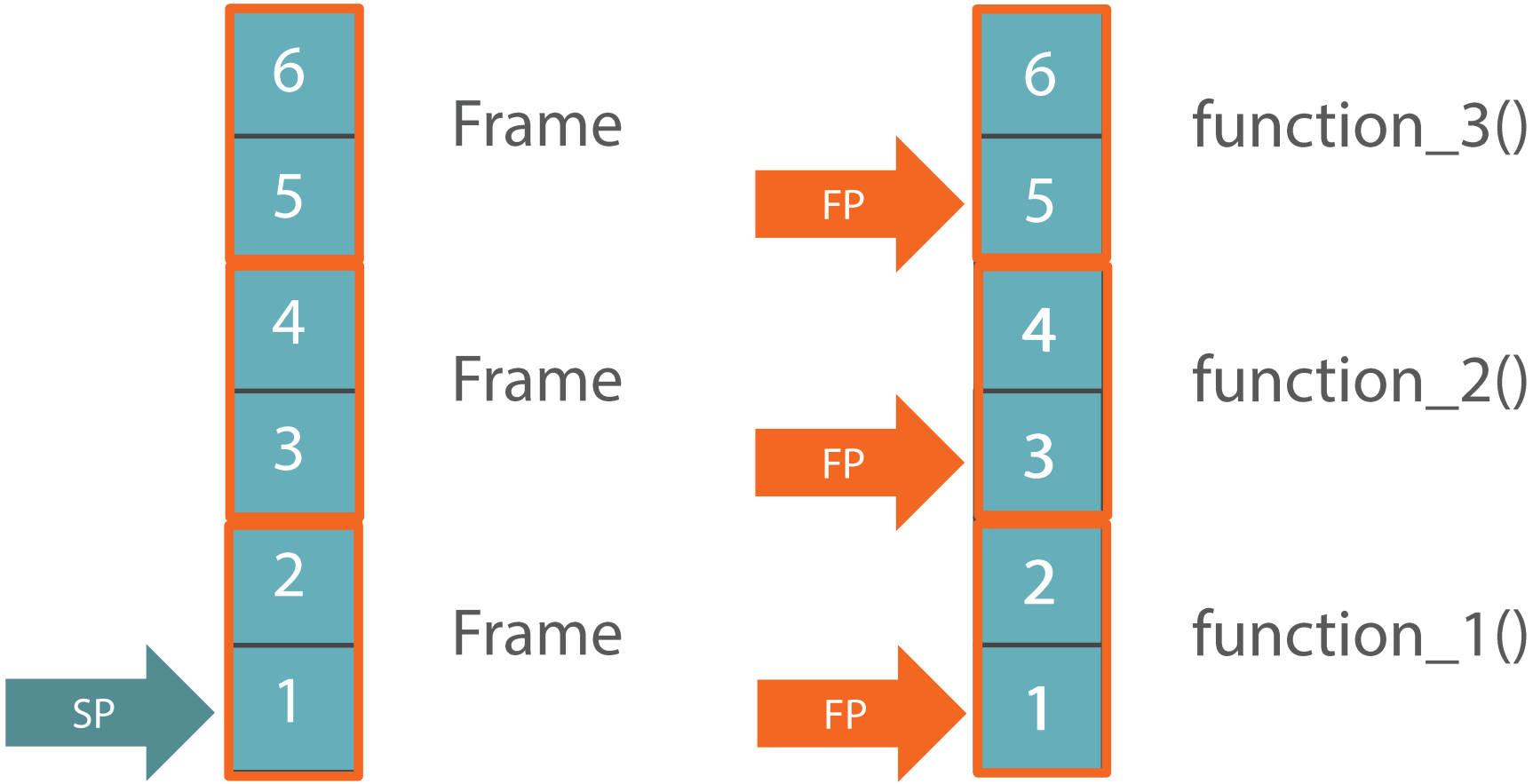
DATA12



DATA12

pluralsight

Stack Frames



The Stack in Code

```
int main()
{
    function_1();
    return 0;
}

int function_1()
{
    int i = 42;
    function_2(i);
    return 1;
}

int function_2(int i)
{
    return 2;
}
```

pluralsight

Inside Stack Frames

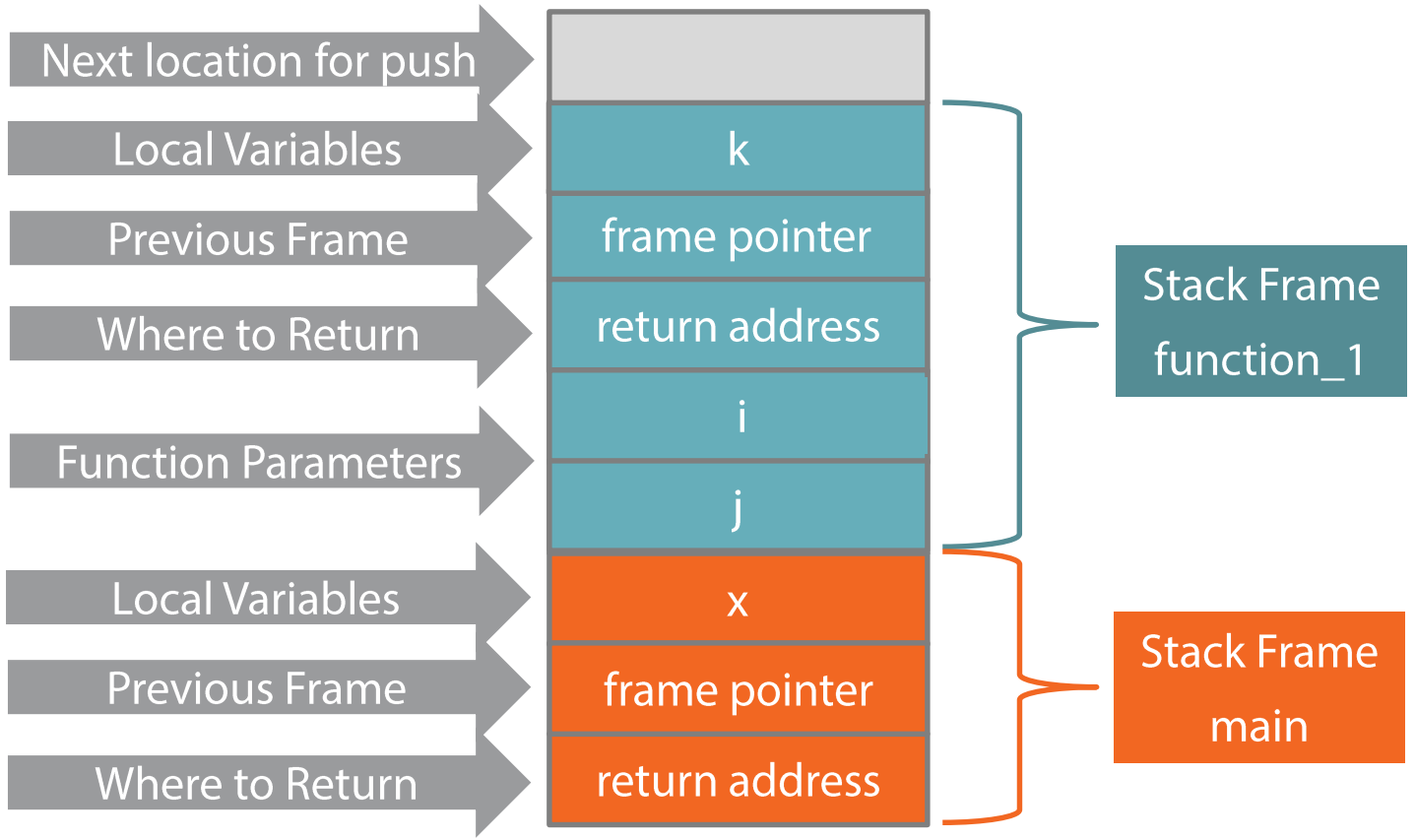
- A stack frame is…
  - …created each time a function is called.
  - …a logical grouping of information associated with a function.
  - …used by the program to find information about the current function.
  - …a way to save function information that must be used later.
  - … why functions do not overwrite the data of other functions.

pluralsight

Inside Stack Frames

```
int main()
{
  int x = function_1(33, 42);
  return 0;
}
```

```
int function_1(int i, int j)
{
  int k = i + j;
  return k;
}
```



Pointers

- A "pointer" is a data variable that stores a memory address
  - a location in the computer's memory
  - C and C++ use pointers to locate and use objects in memory
  - A pointer contains the memory address of the object, not the object itself
  - The address of my house is its location and not the house itself
- The data type of the pointer indicates the type of data stored at that location
  - The data type determines how the memory will be read and written

pluralsight

Stack Pointers

SP
Stack Pointer

FP
Frame Pointer

BP
Base Pointer

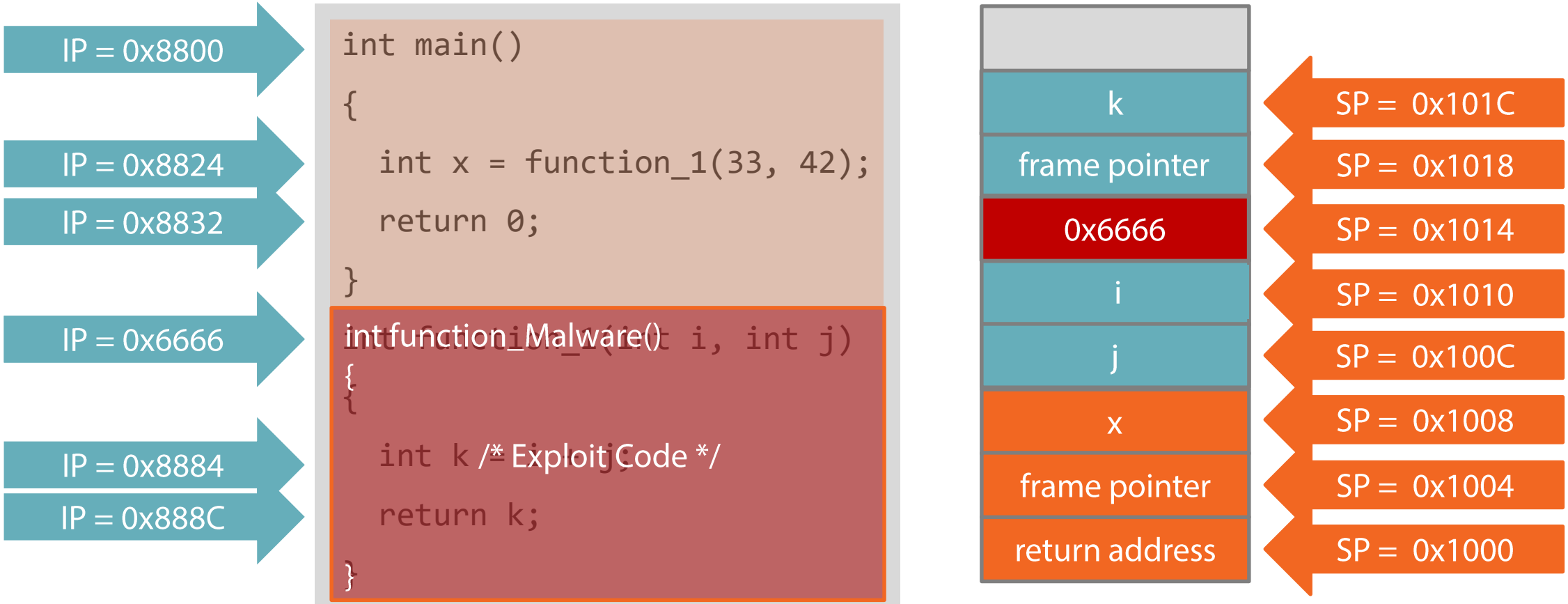
IP
Instruction Pointer

Instruction Counter
or
Program Counter

Registers are dedicated areas of CPU memory

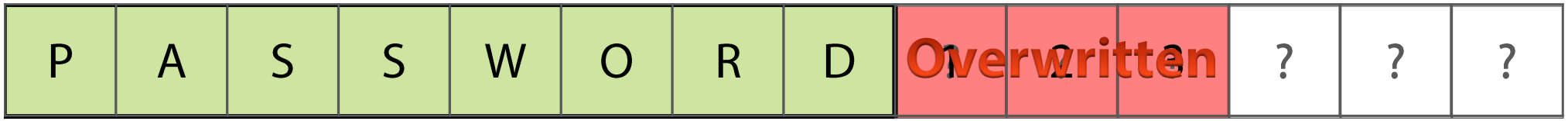
pluralsight

The Instruction Pointer



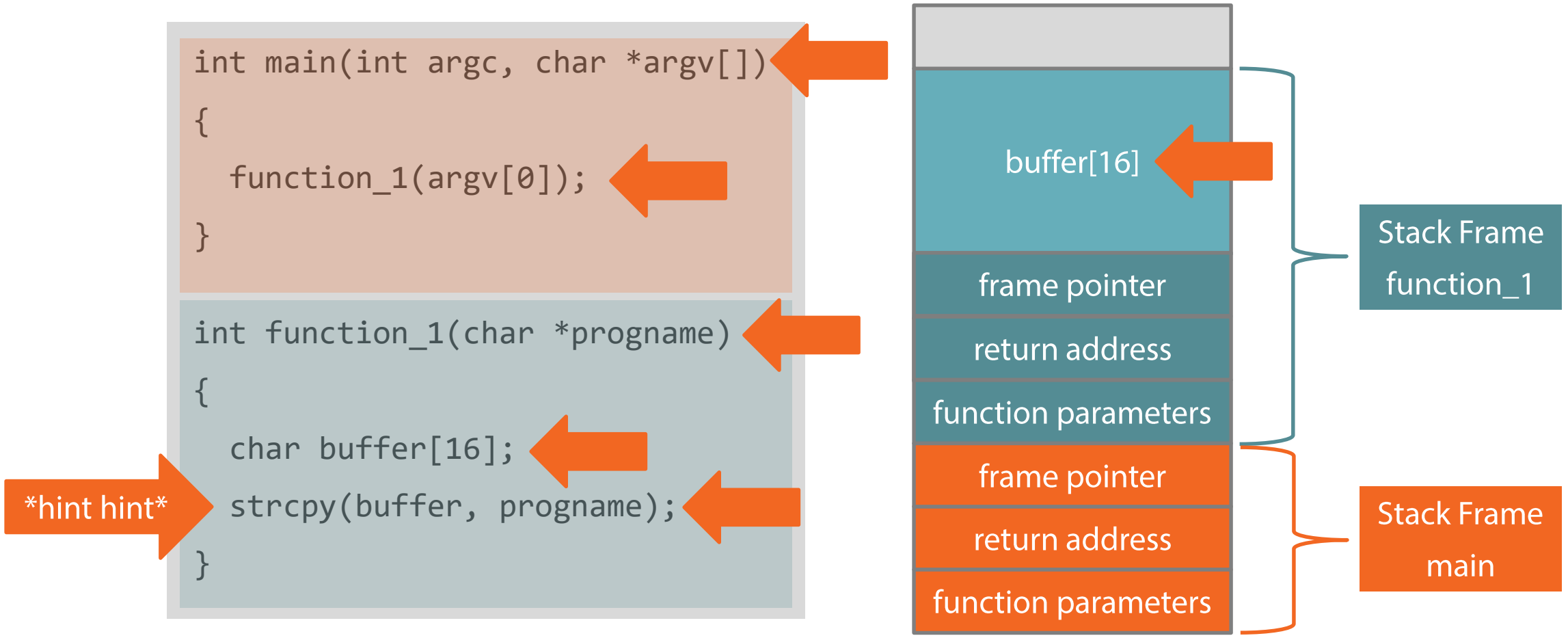
Overflowing the Stack

- An overflow occurs when…
  - data written to a buffer exceeds the boundaries of the buffer
  - the length of the data written is longer than the buffer
  - the extra data is written past the end of the buffer
  - anything following the buffer in memory is overwritten



pluralsight

Overflow Inside the Stack

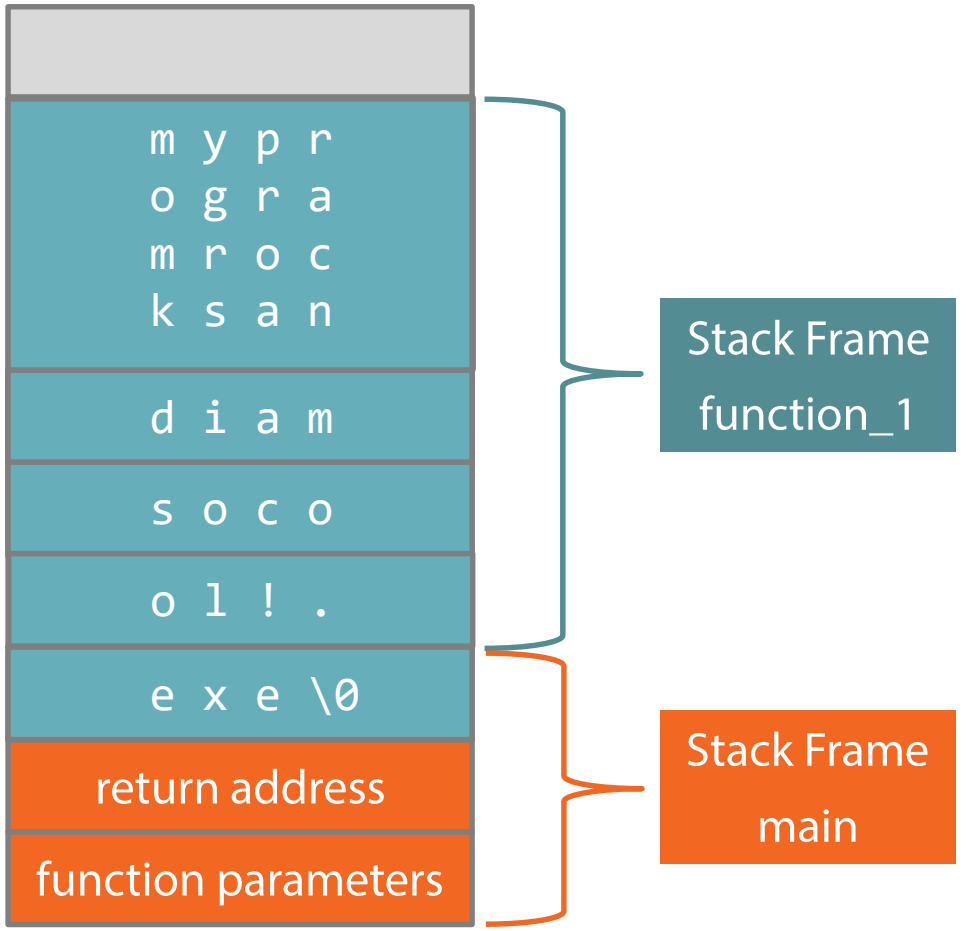


pluralsight

Overflow Inside the Stack

```
> myprogram
> myprogram.exe
> myprogramrocks!
> myprogramrocks!.exe
> myprogramrocksandiamsocool!.exe
```

```
int function_1(char *progname)
{
  char buffer[16];
  strcpy(buffer, progname);
}
```



pluralsight

Exploiting Stack Overflows

What can you do with a buffer overflow condition?

Morris worm

Code Red worm

Many other Malware

Access memory-resident resources

Run installed programs

Inject and run exploit code on your computer

Program instability and abnormal termination

pluralsight

"Exploit Code"

Any executable code that is injected into a program's buffer…

causing the buffer to overflow…

and forcing the program to run the injected code

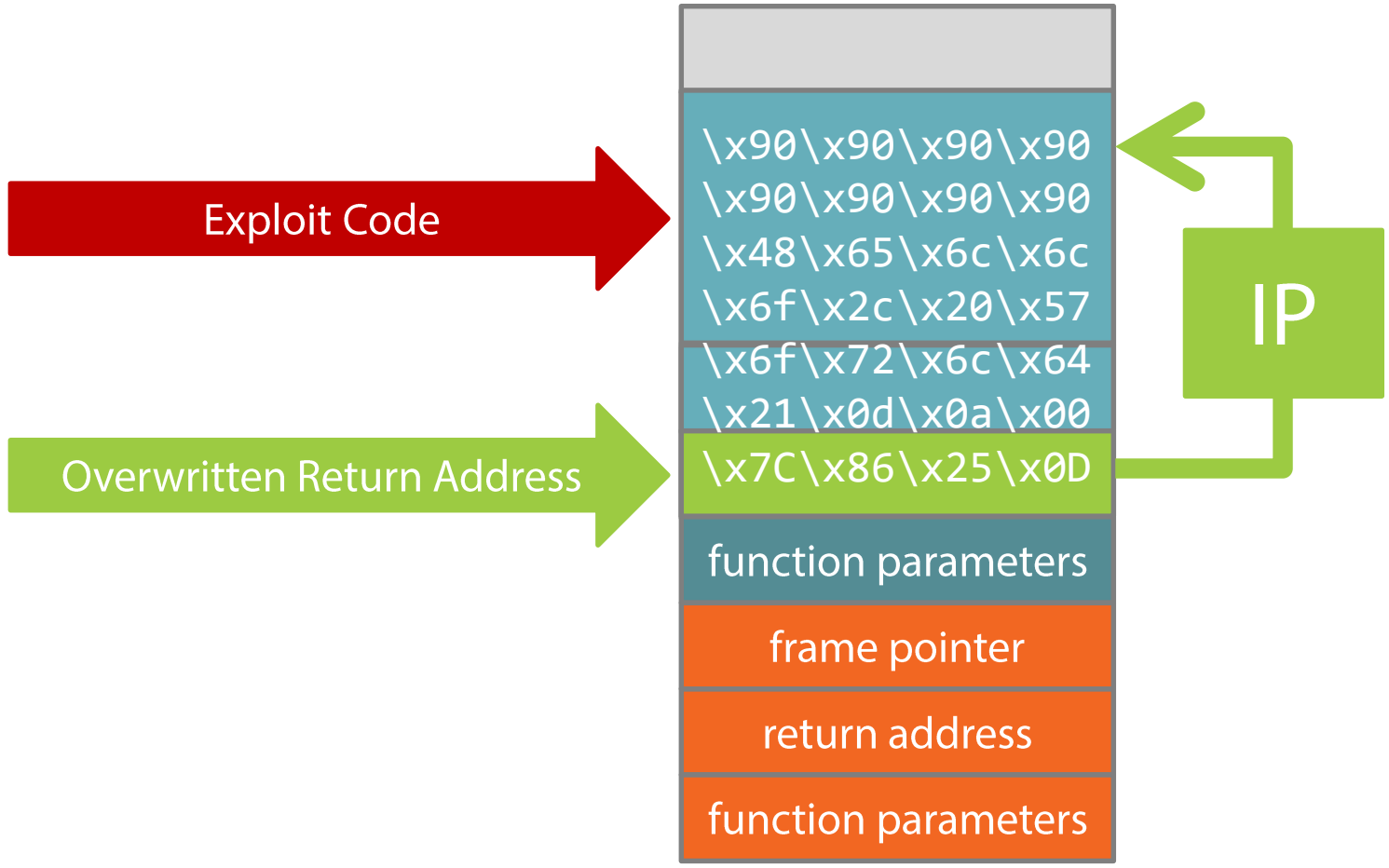
Arbitrary code execution

It doesn't matter what the exploit code does…

…just get it into the system and run it!

pluralsight

Stages of a BoF Exploit



Predictability Is the Key

- Arbitrary code exploits need a predictable execution environment
- The memory location of needed objects must be known or discoverable
- System resources are mapped into the process' virtual memory space
  - 0x7C86250D : kernel32.dll.WinExec()
  - 0x7C801D7B : kernel32.dll.LoadLibraryA()
  - 0x7C802336 : kernel32.dll.CreateProcessW()
  - 0x7C80236B : kernel32.dll.CreateProcessA()
- All processes have the same resources at the same virtual memory locations
- Exploit code needs these system resources to properly run

pluralsight

Where Is My Exploit Code?

The location of the exploit code in the stack or heap must be known

The IP must contain the memory address of the beginning of the exploit code

The exploit code may not be located at a predicable address(es)

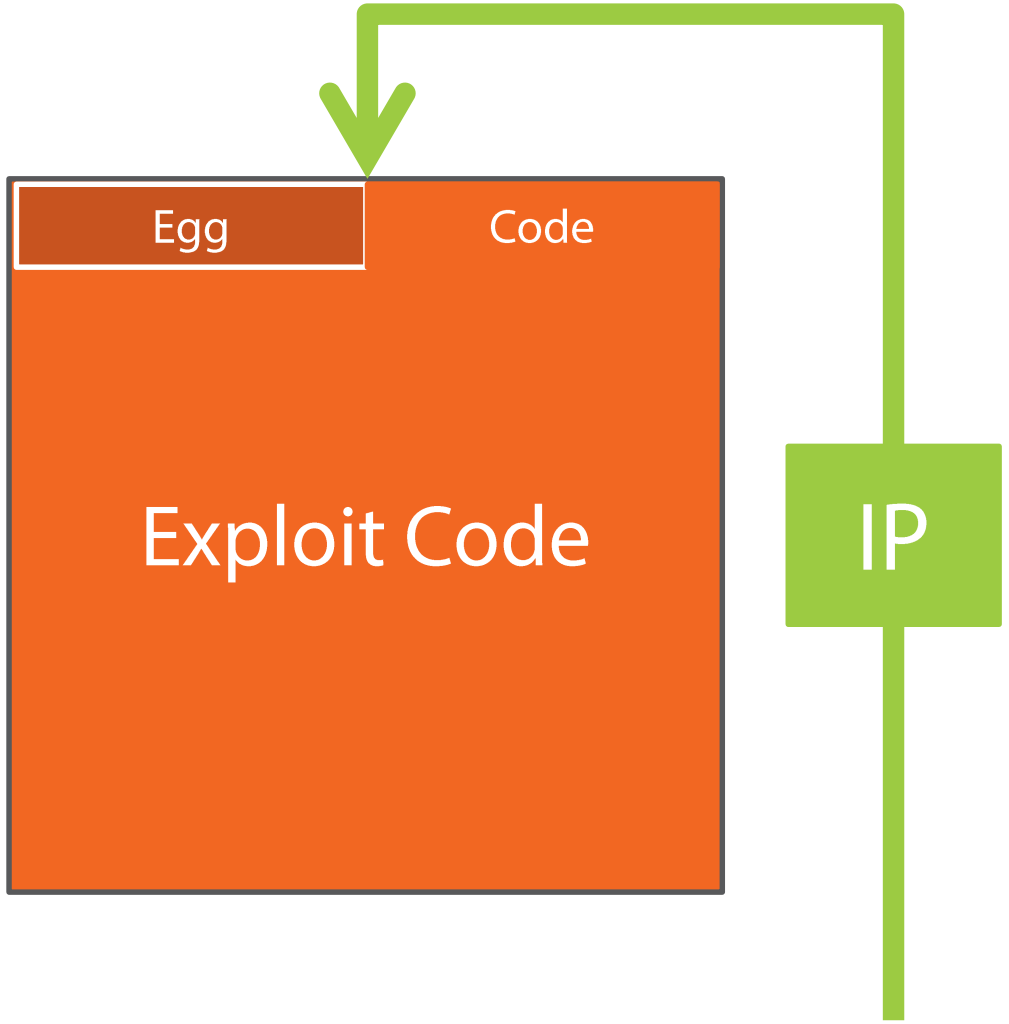
Egghunting

NOP Sled

pluralsight

Egghunting

- The "egg" is a byte signature used to easily find exploit code in memory
- The egg is located at the beginning of the exploit code
- The egg must be unique in process memory
- The first exploit code instruction is located immediately after the egg
- Search code must be pre-loaded and memory exceptions handled



pluralsight

No Operation

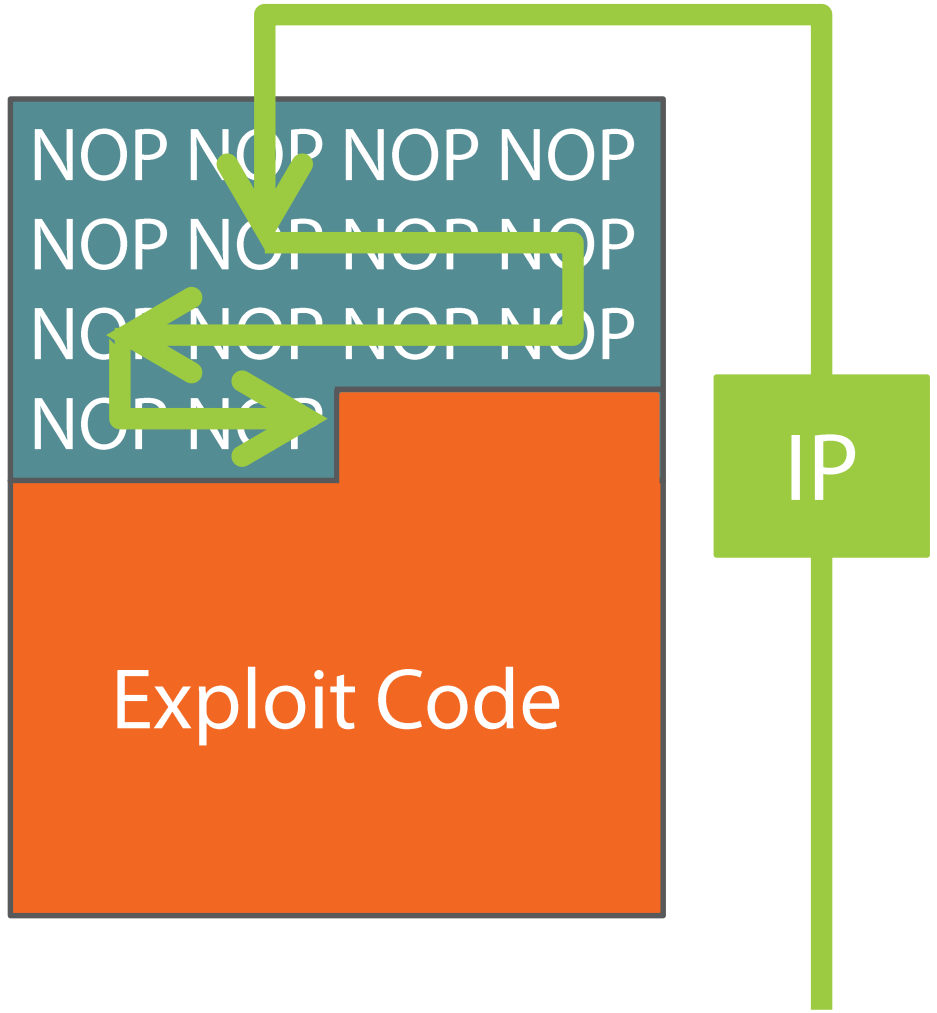
- Instructions command the CPU
- No Operation instruction
- NOP or NO-OP
- Do nothing and advance the IP
- NOP consumes one CPU cycle
- 1 cycle = 10^{-9} seconds on 1 GHz CPU
- NOPs fix timing, synchronization, and code alignment issues

Operations	CPU Instructions
Data Transfer	MOV XCHG BSWAP
Arithmetic	ADD SUB MUL DIV CMP
Logic	AND OR XOR NOT TEST
Stack	POP PUSH ENTER LEAVE
Flow Control	JMP CALL RET INT LOOP
CPU Control	**NOP** HLT PWRSAV RESET

pluralsight

NOP Sled

- Set the IP to the address of the beginning of the exploit code
  - This is not always easy or possible
- The IP only needs to point to the NOP sled
- The IP will then move down to and execute the exploit code
- A NOP sled is a sequence of NOP instructions preceding the payload, making the payload easier to find



pluralsight

Pseudo-NOP Sleds

Long runs of NOP instructions are easily detectable as a NOP sled

Long runs of nonsensical CPU instructions are not (easily) detectable as a NOP sled

pluralsight

Shellcode

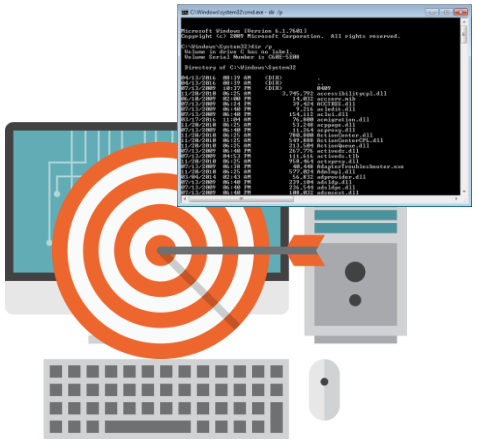
- Shellcode
  - creates an interactive OS shell
  - executes commands on the local system via command line, script, or batch file
  - /bin/sh, /bin/bash, %WINDIR%\System32\cmd.exe, explorer.exe, command.com

- Shellcode can be any type of program
  - Servers (FTP, Telnet, SSH, IRC, HTTP), monitors (sniffers, keyloggers)
- Shellcode runs with the privileges and context of the process that runs it
- Attacker: ”Give me a shell with root user privileges, now!”

pluralsight

Shellcode Types

- Port Binding
- Reverse
- Command Execution Code
- File Transfer
- Find Socket
- Kernel Space
- Multistage
- Process Injection
- System Call Proxy



pluralsight

Creating Shellcode

- Creating shellcode
  1. Writing the shellcode in assembly language (CPU-specific)
  2. Assembling and linking the assembly code to create machine code
  3. Converting the machine code to an escaped-byte text format
- Shellcode code can be written in higher-level languages
- …but small size is important, so stick with assembly language.

```
\xe9\x1e\x00\x00\x00\xb8
\x04\x00\x00\x00\xbb\x01
\x00\x00\x00\x59\xba\x0f
\x00\x00\x00\xcd\x80\xb8
\x01\x00\x00\x00\xbb\x00
\x00\x00\x00\xcd\x80\xe8
\xdd\xff\xff\xff\x48\x65
\x6c\x6c\x6f\x2c\x20\x57
\x6f\x72\x6c\x64\x21\x0d
```

pluralsight

Testing Shellcode

```
unsigned char shellcode[] = “Escaped byte codes";
"\xe9\x1e\x00\x00\x00\xb8\x04\x00\x00\x00\xbb\x01\x00\x00\x00\x
59\xba\x0f\x00\x00\x00\xcd\x80\xb8\x01\x00\x00\x00\xbb\x00\x00\
x00\x00\xcd\x80\xe8\xdd\xff\xff\xff\x48\x65\x6c\x6c\x6f\x2c\x20
\x57\x6f\x72\x6c\x64\x21\x0d\x0a";
int main()
{
    void (*func)();
    func = (void (*)())shellcode;
    (*func)();
}
```

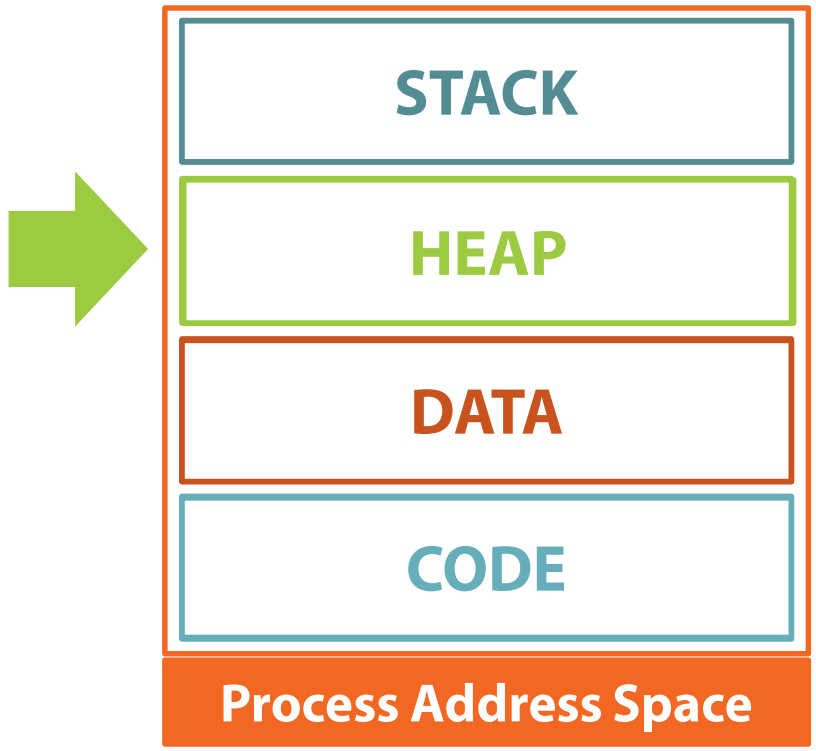
```
int main()
{
    void (*func)();
    func = (void (*)())shellcode;
    (*func)();
}
```

```
gcc -g -Wall -fno-stack-protector -z execstack shelltest.c -o shelltest
```

pluralsight

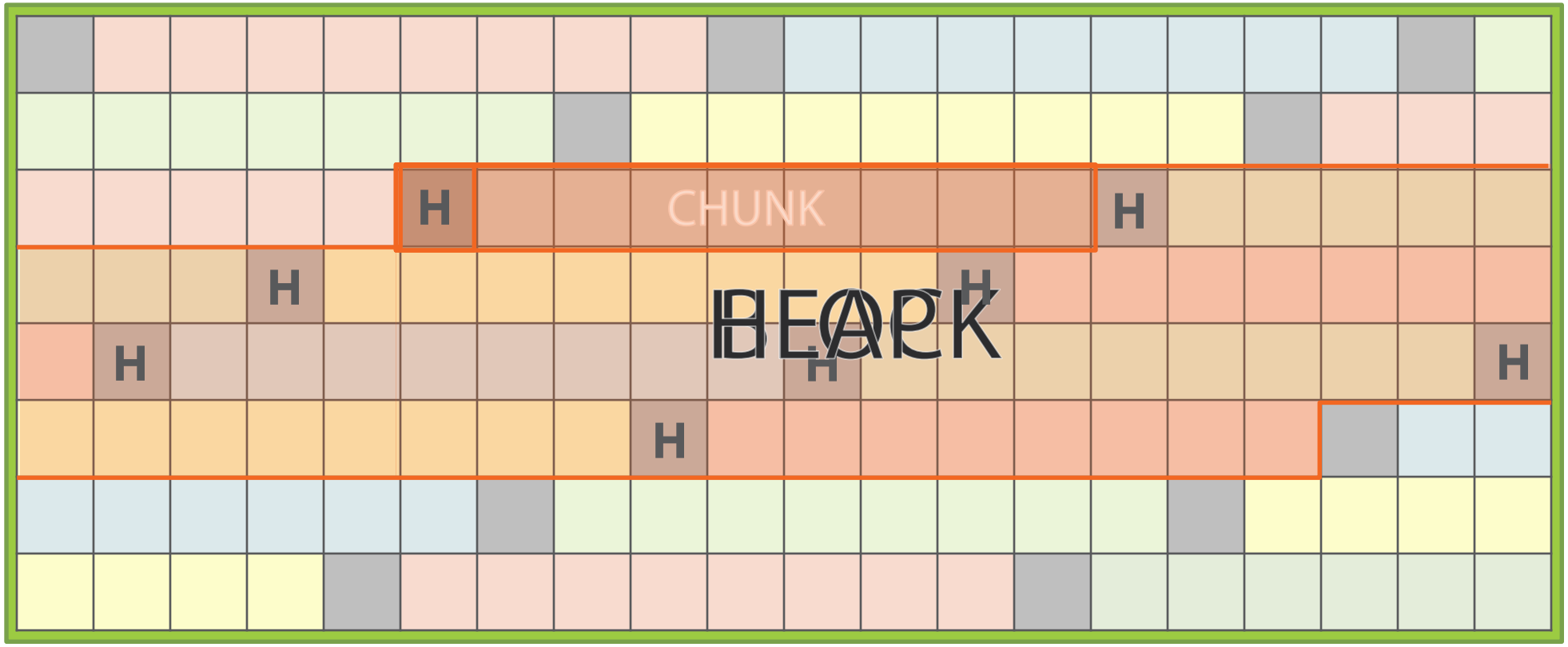
Inside The Heap

- Heap stores program data
- The stack is fixed in size, but the heap can grow and shrink as needed
- Dynamic memory
  - allocates and releases storage space as the process needs
  - used to temporarily store data for processing
- The heap can also store and execute code, which is just a type of data



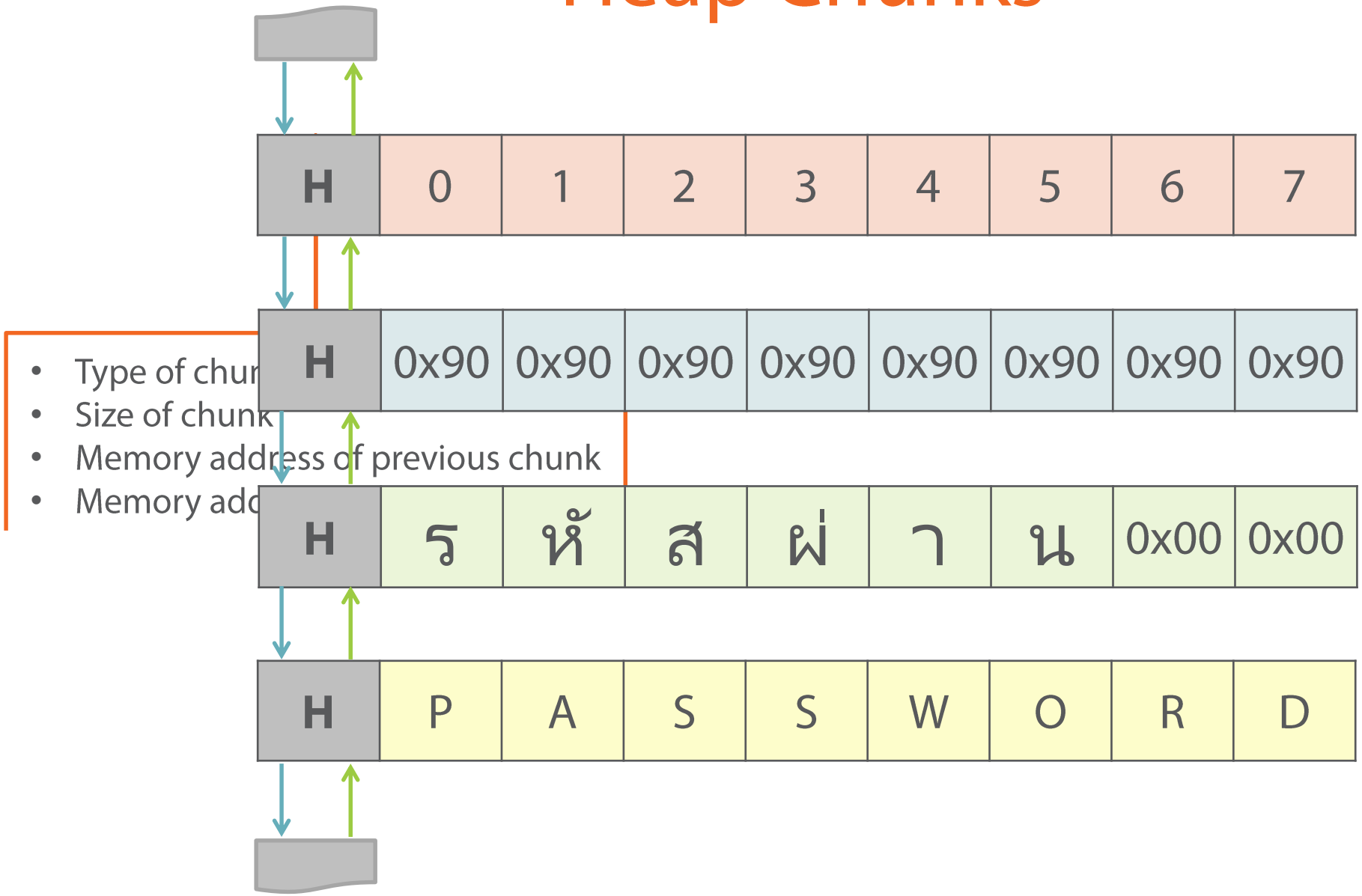
pluralsight

Structure of The Heap



pluralsight

Heap Chunks


H
0
1
2
3
4
5
6
7
H
0x90
0x90
0x90
0x90
0x90
0x90
0x90
0x90
• Type of chur
• Size of chunk
• Memory address of previous chunk
• Memory add
H
ร
หั
ส
ผ่
า
น
0x00
0x00
H
P
A
S
S
W
O
R
D

Using the Heap in Code

```
// Buffer allocated on stack only requires a variable declaration
int bufferOnStack[20];

// Buffer allocated on heap requires a function call
int *bufferOnHeap = (int *)malloc(20);

int* bufferOnHeap2 = new int[20];

int *bufferOnHeap3 = (int *) HeapAlloc(hHeap, 8, 20*sizeof(int));
```

pluralsight

Using the Heap in Code

```
// Releasing buffer on stack only requires exiting function block
{
    int bufferOnStack[20];
}

// Releasing a buffer allocated on heap requires a function call
free(bufferOnHeap);         bufferOnHeap = 0;

delete bufferOnHeap2;       bufferOnHeap2 = 0;

HeapFree(bufferOnHeap3);    bufferOnHeap3 = 0;
```

pluralsight

Exploiting Heap Overflows

- Buffers on the heap can be overflowed just as stack buffers can
- Changes the memory addresses stored in pointer variables
  - Redirect the flow of program execution
  - Cause the process exception handler to be called
  - Crash the running process
- Code stored in the heap can be executed
  - Redirect the Instruction Pointer to the code
  - Find a heap buffer large enough for the exploit code
  - Make "fake" heap memory management structures

pluralsight

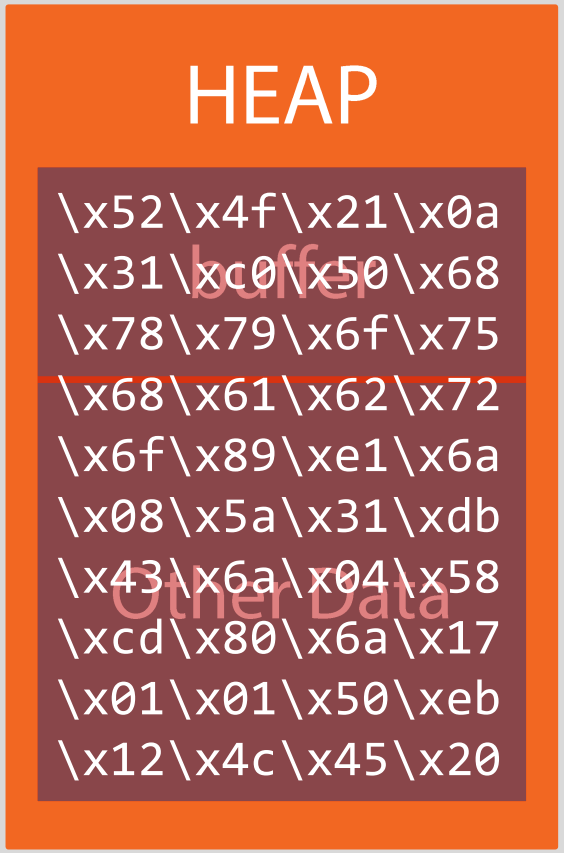
Is a Buffer on the Stack or Heap?

- Important to know for writing BoF
- Look in the source code or use static/dynamic analysis
- How big is the buffer and what is it used for?
- Heap buffers are needed when:
  - very large data must be stored
  - the buffer must exist for a long time
  - the size of the buffer needs to change
  - the size of the buffer is only known at run-time

pluralsight

Heap Overflow in Code

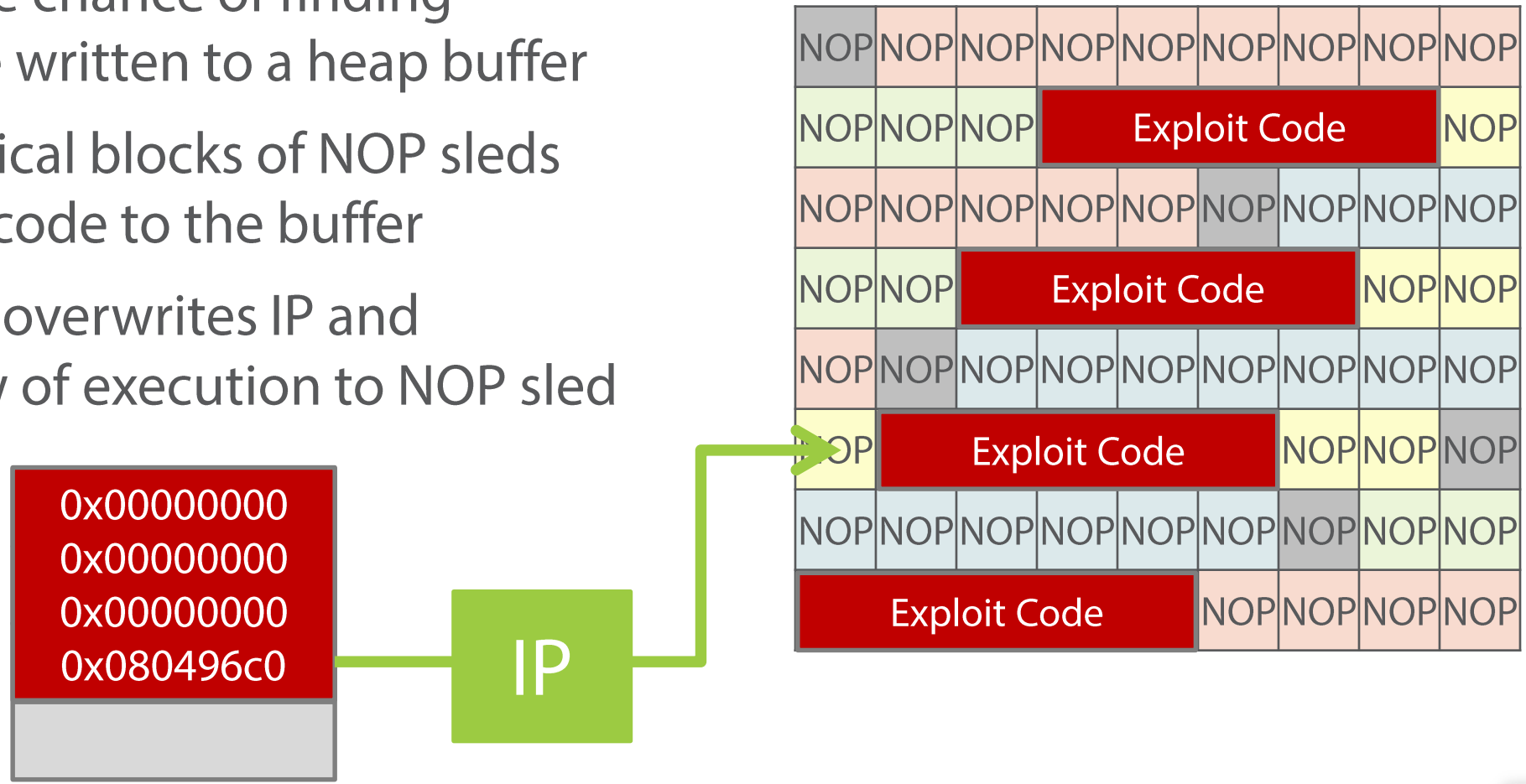
```
int main(int argc, char *argv[])
{
    char *buffer = (char*)malloc(16);
    strcpy(buffer, argv[1]);
    free(buffer);
}
```



pluralsight

Heap Spray

- Increases the chance of finding exploit code written to a heap buffer
- Writes identical blocks of NOP sleds and exploit code to the buffer
- BoF in stack overwrites IP and redirect flow of execution to NOP sled in buffer



pluralsight

Function Pointers

A variable that stores the location of a function

Functions are usually called by name in code

But which function to call in what situation?

Call a function by its memory address, not by its name

Decide which function to call while a program is running

Functions called by name must be hard-coded when the software is written

pluralsight

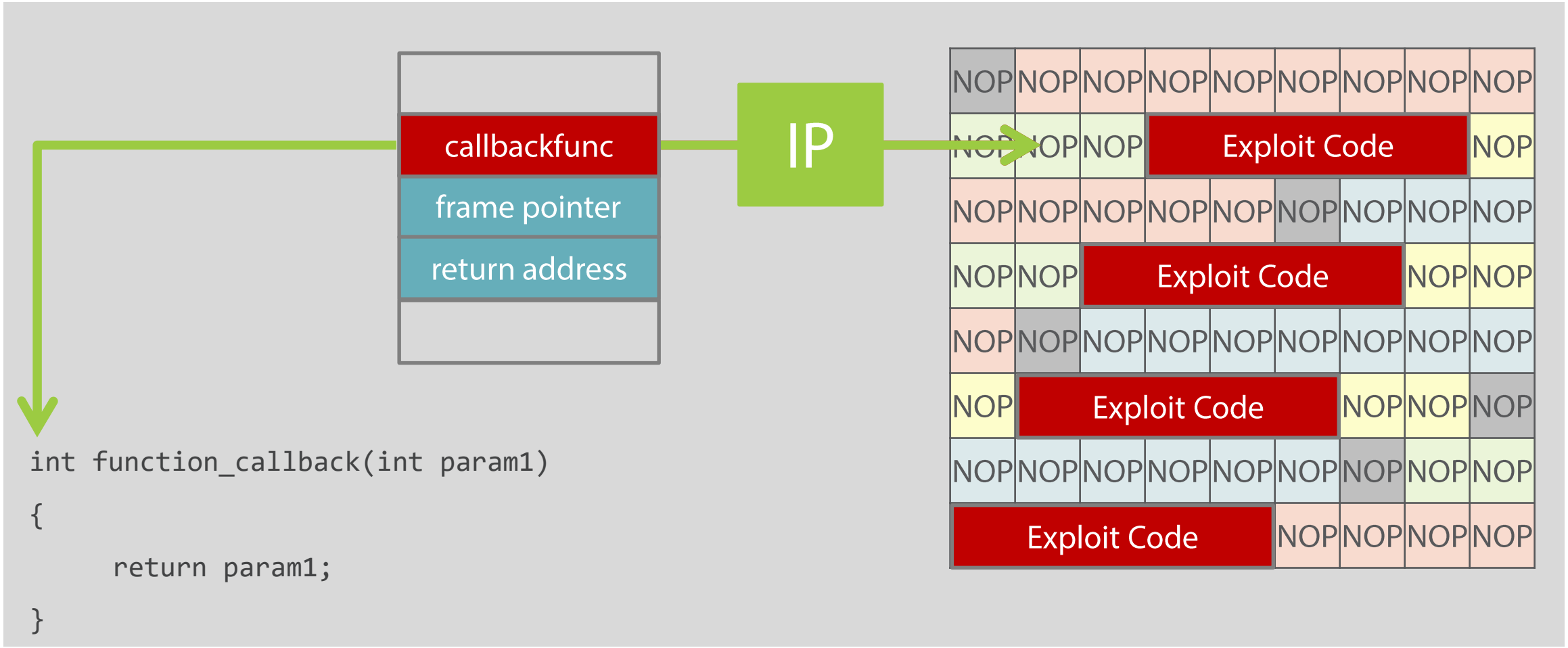
Function Pointers in Code

```
int main()
{
    int (*callbackfunc)(int);
    callbackfunc = (int (*)())function_callback;
    int result = (*callbackfunc)(42);
}
int function_callback(int param1)
{
    return param1;
}
```




pluralsight

Function Pointer Overwrite



pluralsight

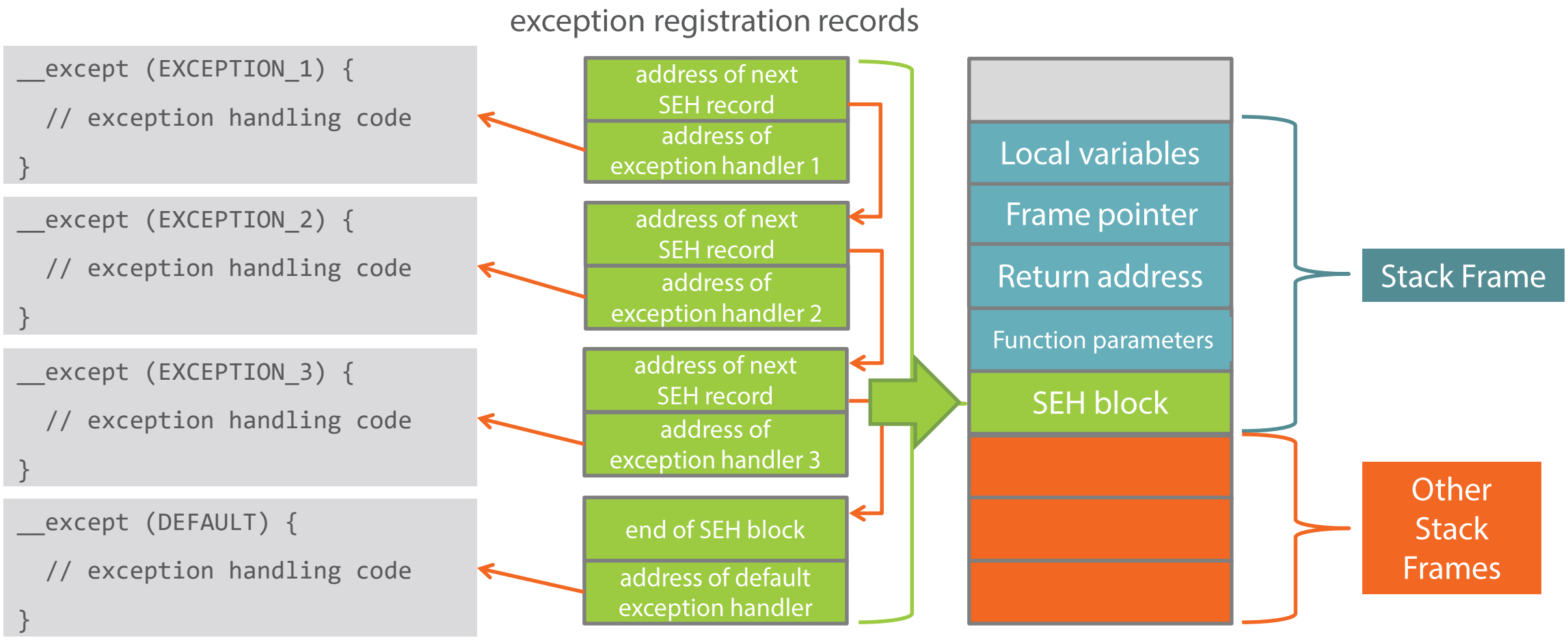
Structured Exception Handling

- Windows OS error recovery feature
- Notification of the occurrence of error conditions
- Allows process to handle errors rather than crashing
- __try, __except, __leave, __finally, RaiseException
- All Window processes have SEH, not just for those written in C++

```
int main()
{
    __try
    {
        TestExceptions();
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
        puts("Executing SEH __except block");
    }
    return 0;
}
```

pluralsight

SEH in the Stack



Exploiting SEH

BoF authors have software problems to solve too

How to execute exploit code after it is injected?

How to keep exploit code from crashing the process?

Use SEH!

pluralsight

CodeRed's SEH Exploit

- Exploited buffer overflow in Microsoft Index Server used by Microsoft Internet Information Server (MS01-033)
- GET/default.ida?NNN%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0
- Overwrites Index Server's SEH records in stack
- Triggering exception causes Windows to execute exploit code in the stack pointed to by the overwritten SEH record

pluralsight

Inside Integer Overflows

Integer overflows are arithmetic errors that occur

when adding or multiplying two or more integers

results in a value too large to store

An unexpected value will result

If integer overflows are not detected and corrected

problems such as buffer overflows could result

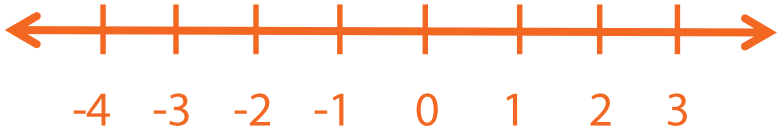
pluralsight

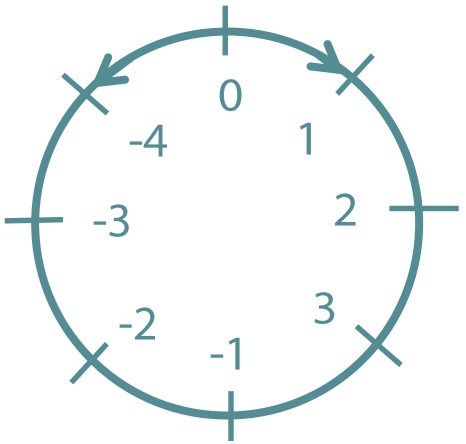
Integer Sizes

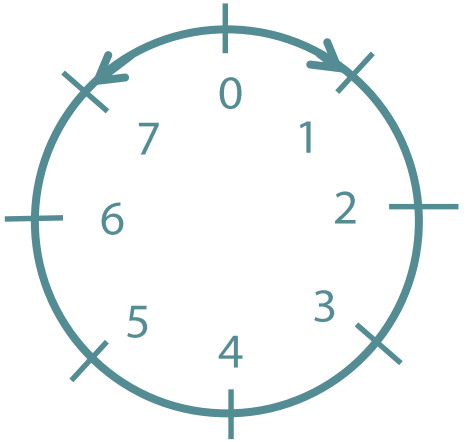
Variable Data Type	Size	Range
signed char	8 bits	-128 to 127
unsigned char	8 bits	0 to 255
signed short int	16 bits	–32,768 to 32,767
unsigned short int	16 bits	0 to 65,535
signed long int	32 bits	–2,147,483,648 to 2,147,483,647
unsigned long int	32 bits	0 to 4,294,967,295
signed long long int	64 bits	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long int	64 bits	0 to 18,446,744,073,709,551,615

pluralsight

It’s Loops, Not Infinity







Overflowing an Integer

```
// unsigned char stores 0 to 255
unsigned char value = 255;
unsigned char result = value + 1;
// result will equal zero, not 256

// signed char stores -128 to 127
signed char value = -128;
signed char result = value - 1;
// result will equal 127, not -129
```

Problem or Business As Usual?

Integer overflows are normal and to be expected

Programmers must catch and correct in their code

Overflows values result in bad program decisions

Reduce your attack surface and patch what remains

pluralsight

Summary

- ✓What's inside of a computer program?
- ✓How code become executable programs
- ✓What happens when program is executed
- ✓Organization of process memory

pluralsight

Summary

- ✓What the stack is and how it works
- ✓What happens when a stack overflow occurs
- ✓How stack overflows are exploited
- ✓What is shellcode, egghunting, NOP sleds

pluralsight

Summary

- ✓What is heap memory
- ✓How heap buffers can be overflowed
- ✓How heap overflows can be exploited
- ✓Heap spray technique

pluralsight

Summary

- ✓How function pointers can be exploited
- ✓Structured Exception Handling (SEH)
- ✓Integer overflows

pluralsight

The last good thing written in C was Franz Schubert's Symphony number 9.

— Erwin Dieterich

pluralsight

Next Up:

Finding Buffer Overflows

pluralsight