Buffer Overflow Attacks

Buffers and Buffer Overflow

- Buffer: Temporary area for data storage
- Buffer Overflow: When the application writing to a buffer, it overruns the buffer’s area and writes to memory beyond the buffer
- ~ Attack: When buffer overflow is done on purpose to hijack code execution (when the buffer is on the stack, it is also called “stack smashing”)

Buffer Overflow - Example

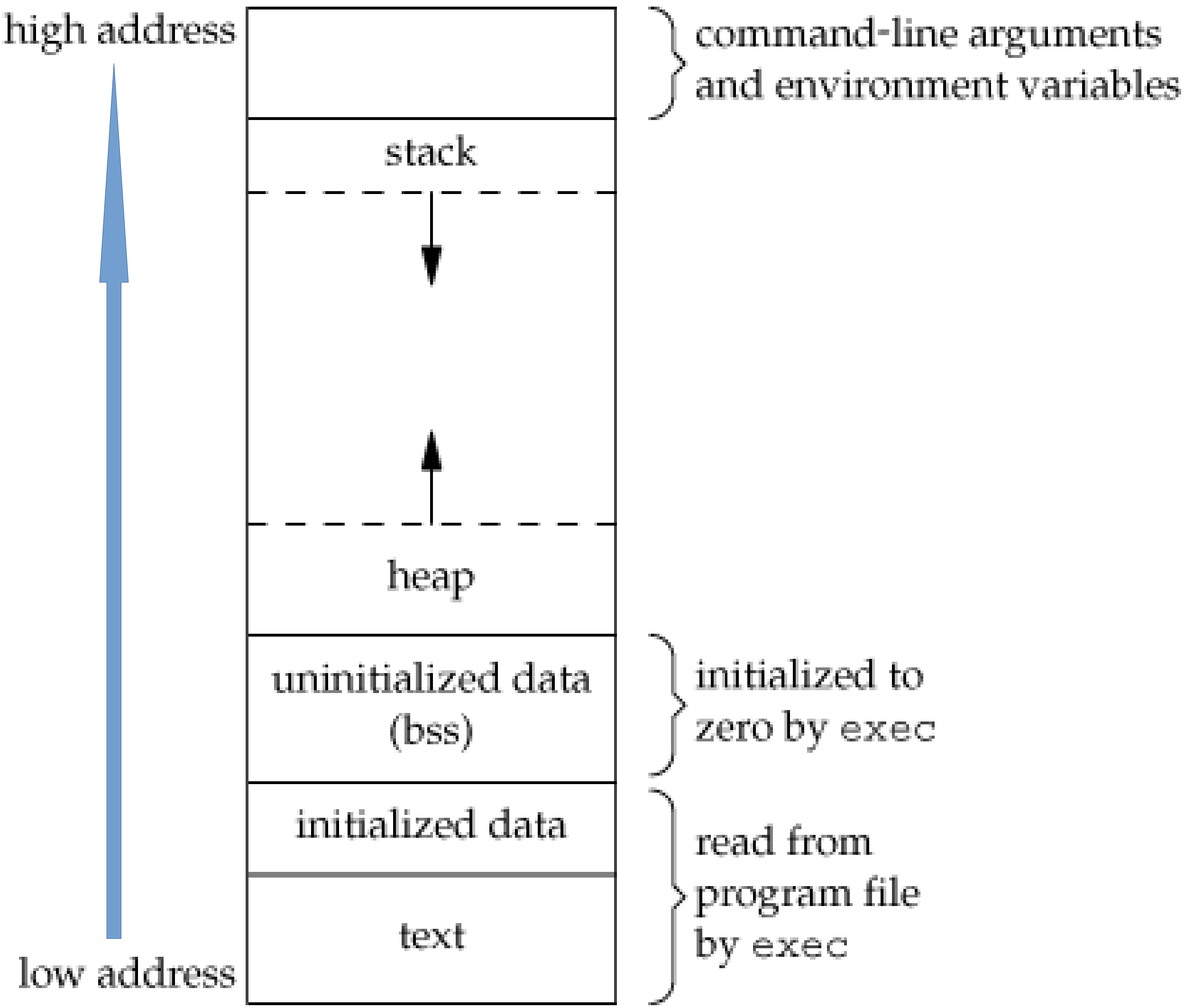
Variable name	A								B	
value	[uninitialised string]								65535	
Hex value									FF	FF

After inserting the string “tooolarge”:

Variable name	A								B	
value	‘t’	‘o’	‘o’	‘o’	‘l’	‘a’	‘r’	‘g’	101	
Hex value	74	6f	6f	6f	6c	61	72	67	65	00

Memory Allocation Types

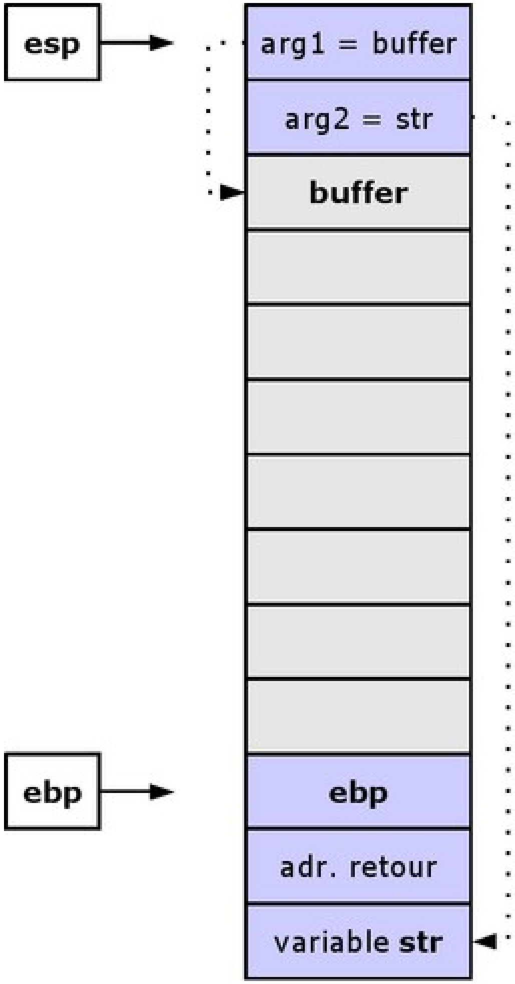
- Static
- Dynamic
  - On the stack
  - On the heap


high address
low address
command-line arguments and environment variables
stack
heap
uninitialized data (bss)
initialized to zero by exec
initialized data
text
read from program file by exec

Source: https://www.geeksforgeeks.org/memory-layout-of-c-program/

Three Special Pointers

- eip: extended instruction pointer (x86) – pointer to the next instruction to execute
- ebp: the base pointer (x86) - A pointer that is used to reference all the function parameters and local variables in the current stack frame
- esp: the stack pointer (x86) – pointer to the next free space in the stack



(Source: wikipedia)

Types of Buffer Overflow

Depends on the location in program memory:

- Stack Buffer Overflow
- Heap Buffer Overflow

Heap Buffer Overflow

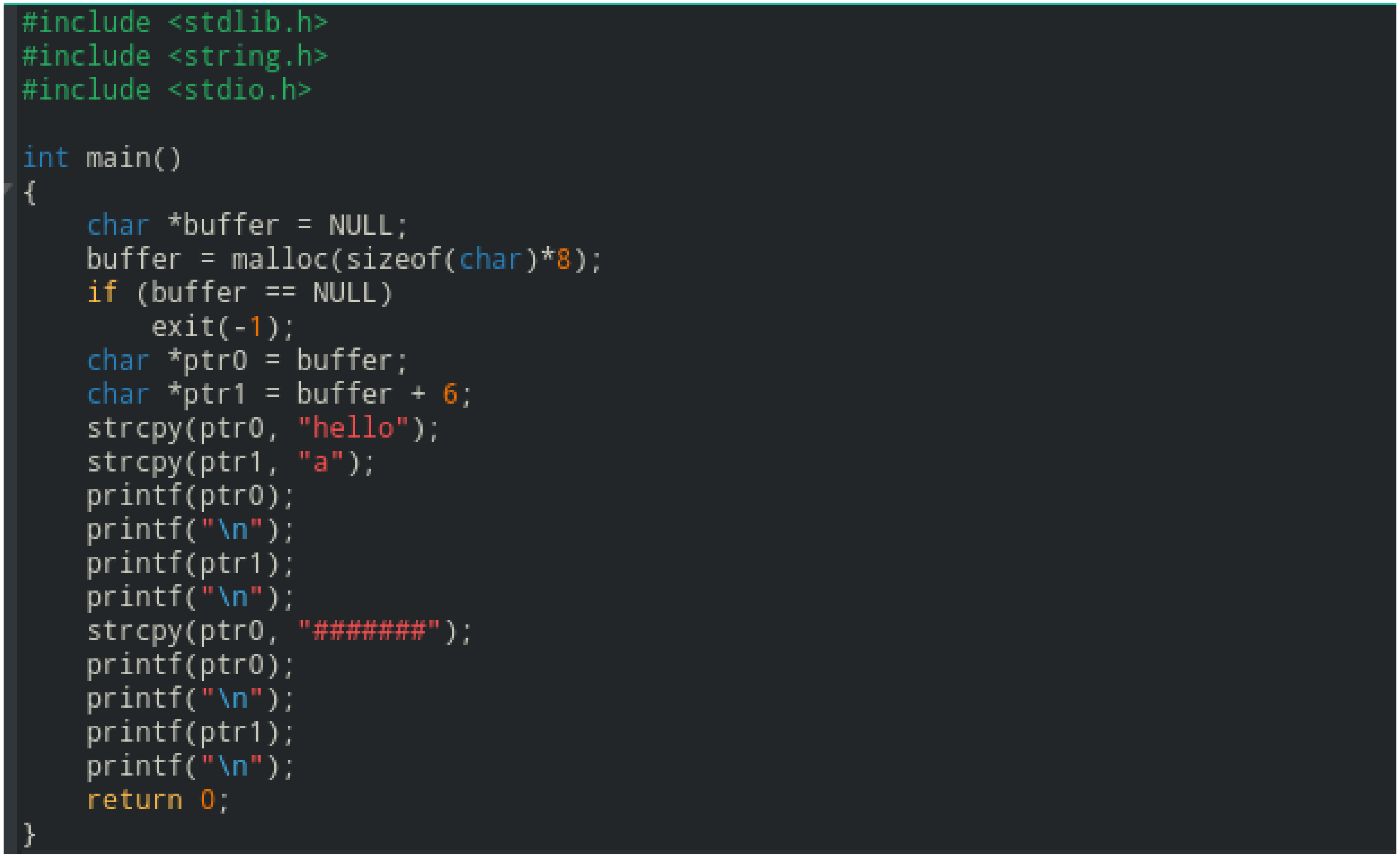
- The less common one of the two
- This type of memory is dynamically allocated
- Exploits using this type usually overwrite internal structures, eg. linked list pointers

Heap Buffer Overflow - Example

	0xA0 … 0xA7		...		0xB0 … 0xB7
	Array a[8]		Unknown Data		Array b[8]

```
strcpy(a, “someoverly…. The end”)
```

Heap Buffer Overflow - Example



```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main()
{
    char *buffer = NULL;
    buffer = malloc(sizeof(char)*8);
    if (buffer == NULL)
        exit(-1);
    char *ptr0 = buffer;
    char *ptr1 = buffer + 6;
    strcpy(ptr0, "hello");
    strcpy(ptr1, "a");
    printf(ptr0);
    printf("\n");
    printf(ptr1);
    printf("\n");
    strcpy(ptr0, "#######");
    printf(ptr0);
    printf("\n");
    printf(ptr1);
    printf("\n");
    return 0;
}
```

Heap Buffer Overflow – Security Measures

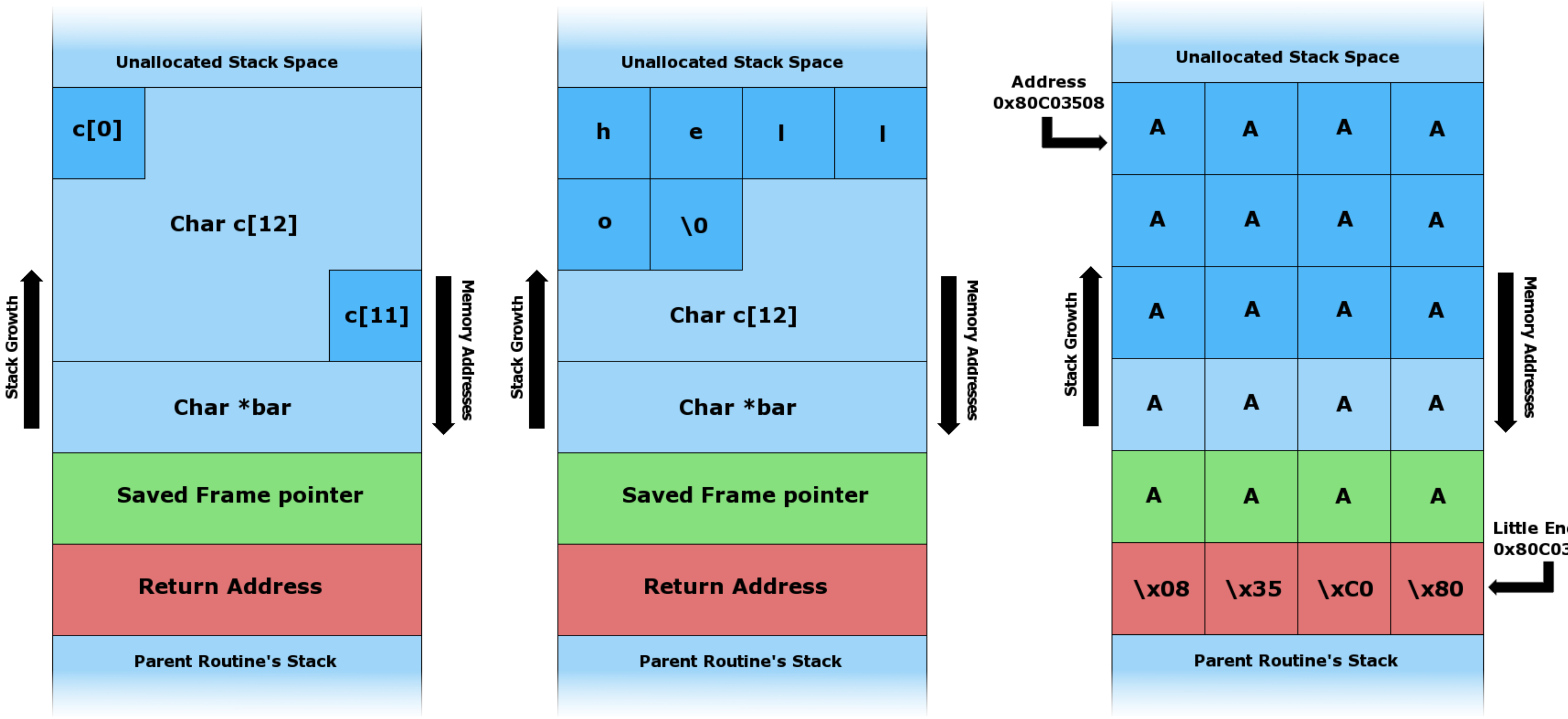
- NX-Bit
- Address Space Layout Randomization
- Sanity Checks

PaX has a robust implementation of ASLR

Stack Buffer Overflow

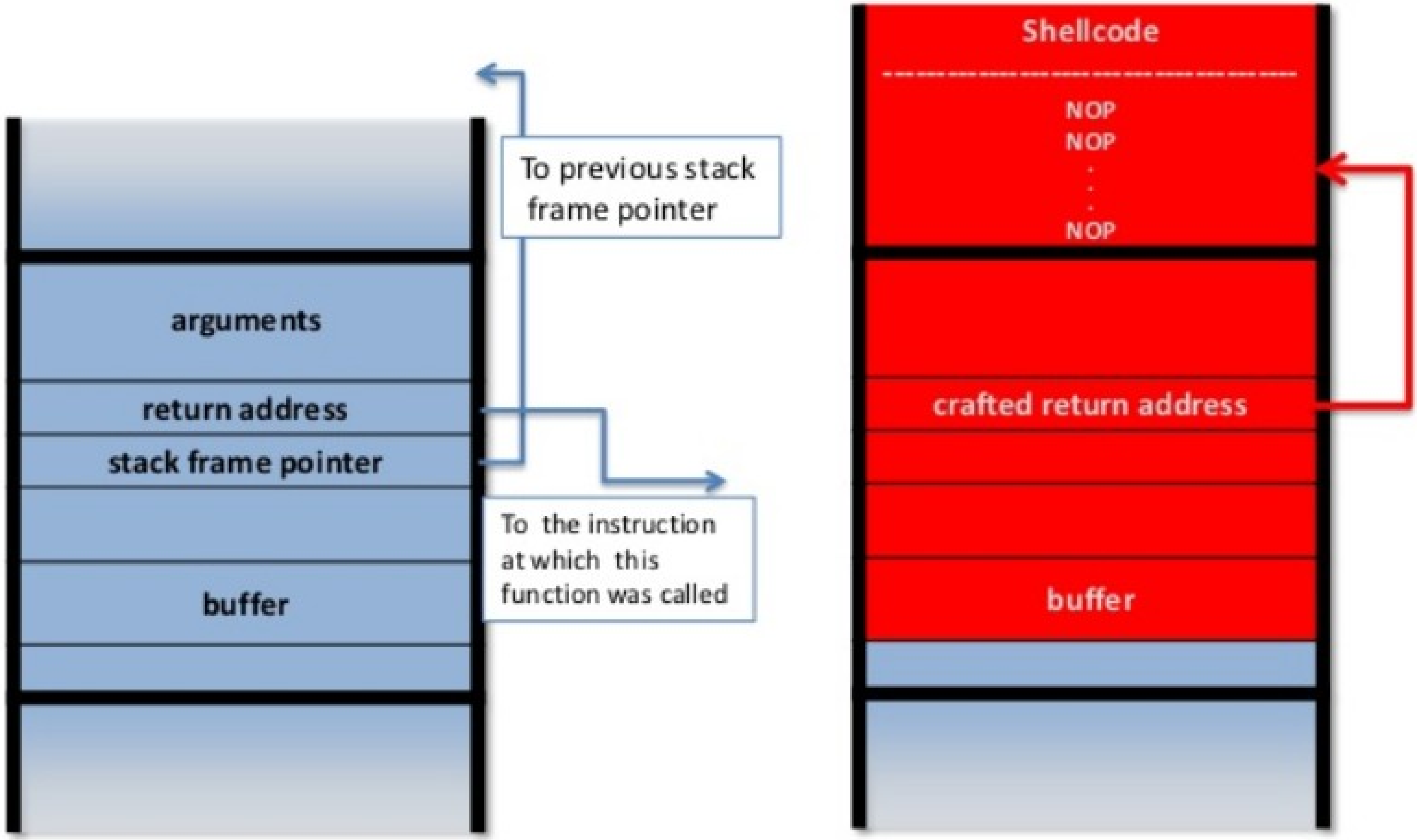
- Occurs when a program writes outside the intended data structure in the call stack
- Results in corruption of adjacent data on the stack
- More likely to derail code execution than overflow on the heap → better for exploitation

Stack Buffer Overflow – Memory View



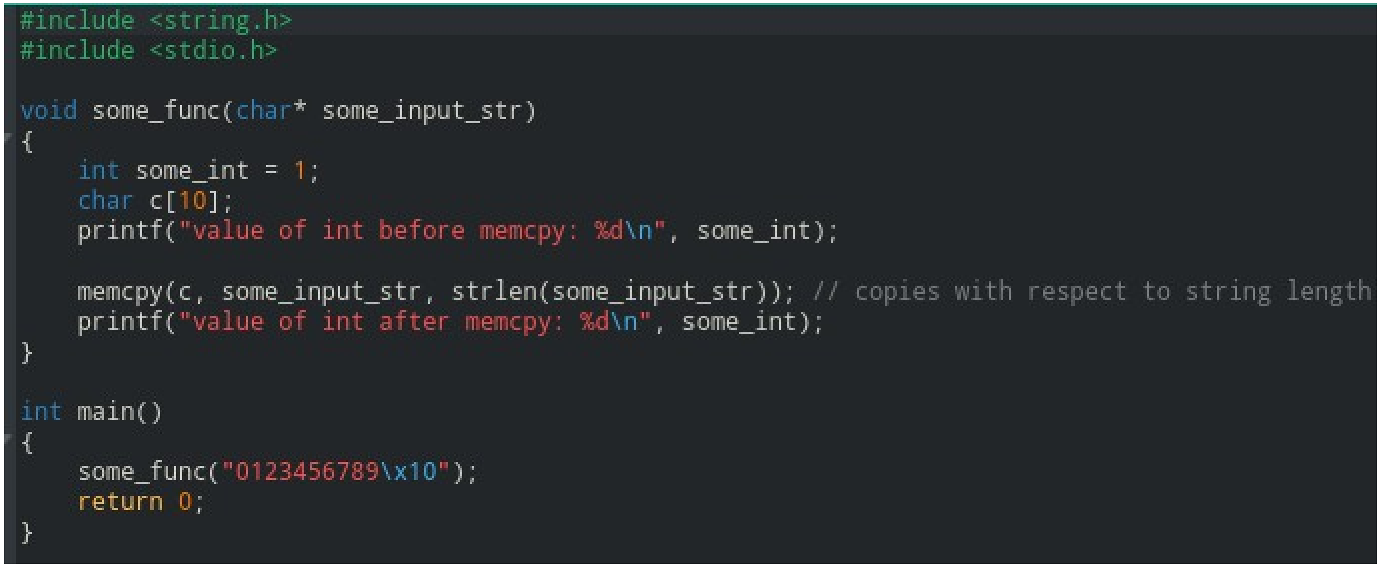
(from Wikipedia)

NOP Slide



Source: Stephanie Rogers – slideshare.net

Stack Buffer Overflow – Example



```c
#include <string.h>
#include <stdio.h>

void some_func(char* some_input_str)
{
    int some_int = 1;
    char c[10];
    printf("value of int before memcpy: %d\n", some_int);

    memcpy(c, some_input_str, strlen(some_input_str)); // copies with respect to string length
    printf("value of int after memcpy: %d\n", some_int);
}

int main()
{
    some_func("0123456789\x10");
    return 0;
}
```

Memory Address

Variable name	c								some_int	
value	‘0’	‘1’	‘2’	‘3’	‘4’	…	‘8’	‘9’	101	
Hex value	30	31	32	33	34	…	38	39	10	01

Stack Buffer Overflow – Protection Measures

- Programming Language Protection
- Safe Libraries
- Executable Space Protection (NX-Bit)
- Address Space Layout Randomization
- Deep Packet Inspection
- Compiler-based protection (Canary, Pointer Protection)

Programming Language Protection

- Languages that don’t allow direct access to memory & strongly typed are generally safe (Java, COBOL, Python etc.)
- Runtime checking and compile-time checking for data being overwritten (Rust, D, Lisp etc.)

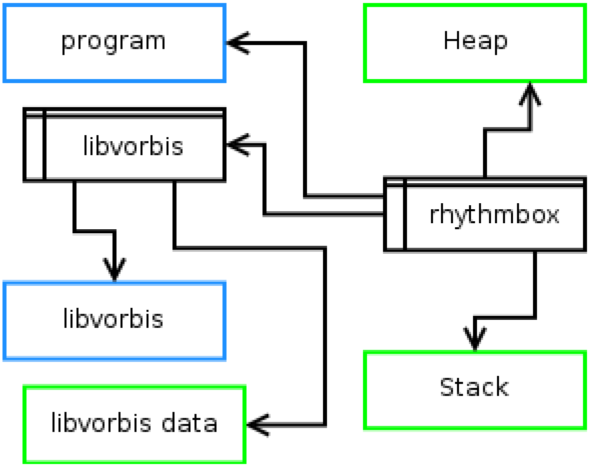
Safe Libraries

- In C and C++, standard libraries are low level
- They let the programmer manage memory directly → often no bound checks (scanf, gets, strcpy)
- Examples of libraries for C and C++ that are safer: Better String Library, Vstr, Erwin

http://bstring.sourceforge.net/
http://www.and.org/vstr/
http://www.theiling.de/projects/erwin.html

Executable Space Protection

- Separation of code and data by marking memory locations as ”executable”
- When execution is rerouted to an arbitrary piece of code in RAM, it will raise a CPU exception
- Examples: Pax, Exec Shield, Openwall

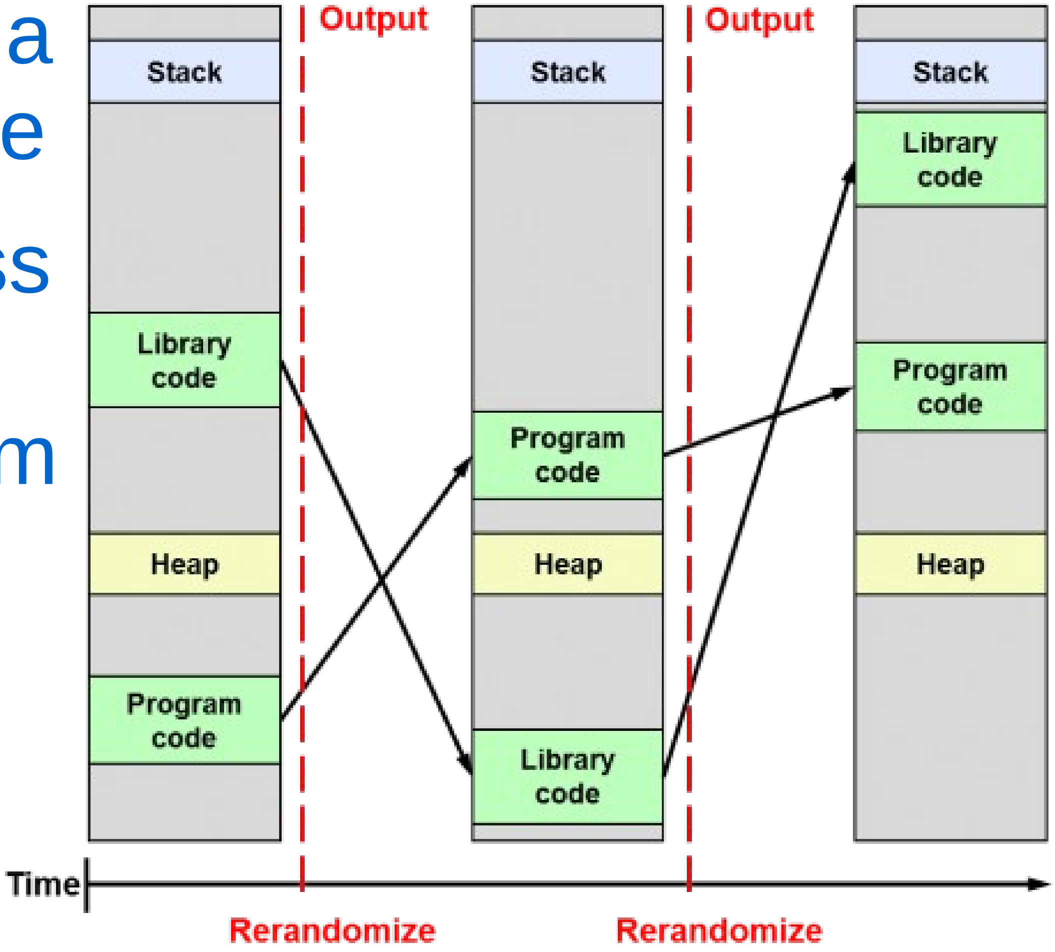


blue = code
green = data
(from wikipedia)

Openwall
bringing security into
open environments

Address Space Layout Randomisation (ASLR)

- Arranges the positions of key data areas (executable's base, library, heap and stack positions) randomly in a process' address space
- Virtual memory address randomisation forces to adapt to each system



(figure source: http://www.daniloaz.com/en/differences-between-aslr-kaslr-and-karl/)

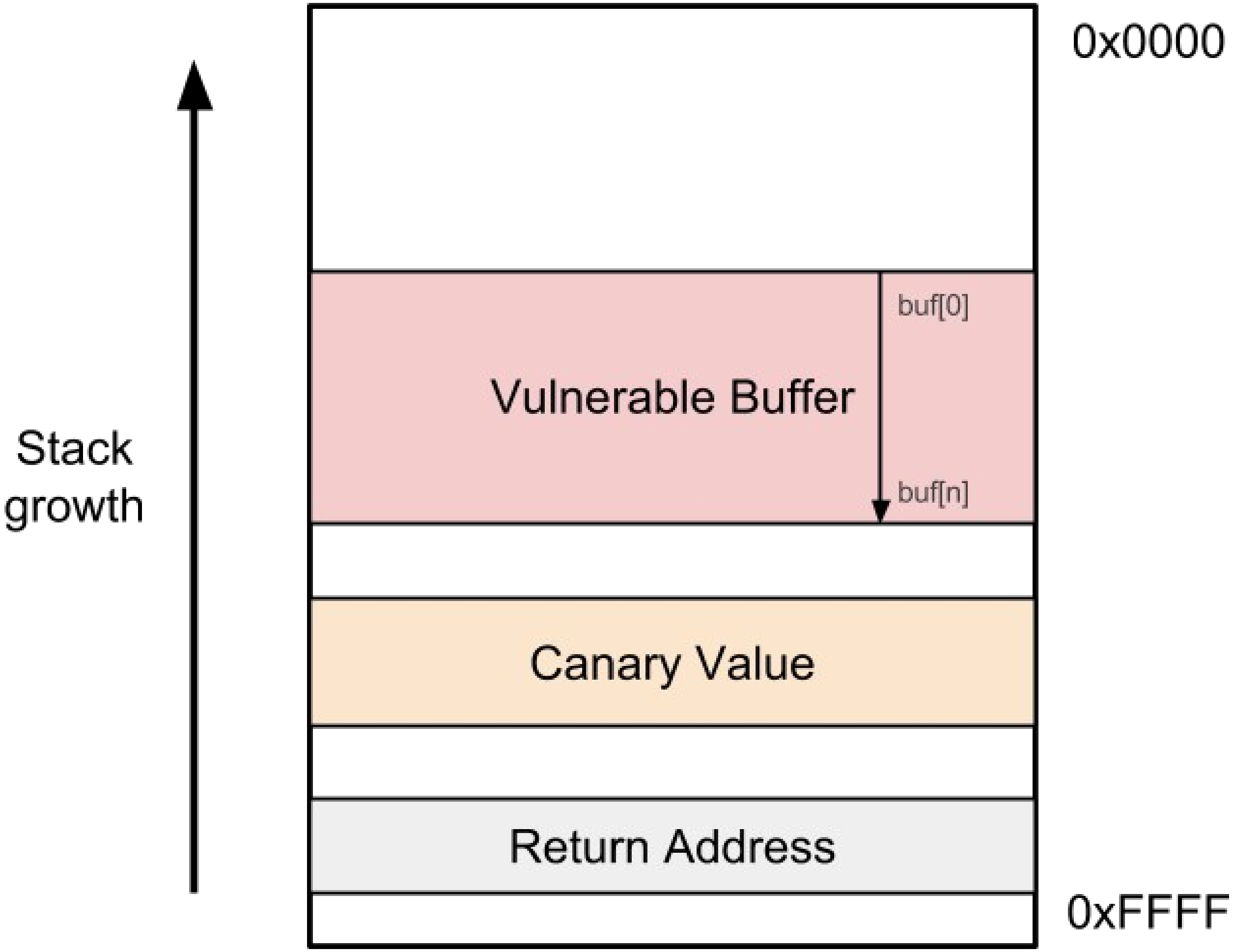
Deep Packet Inspection

- Inspects data being sent over a network
- Blocks/Reroutes/Logs
- Detects attacks by attack signatures and heuristics
- Not effective since attack patterns can easily be randomised (eg. NOP ladder randomization)

Compiler Based Protection - Canary

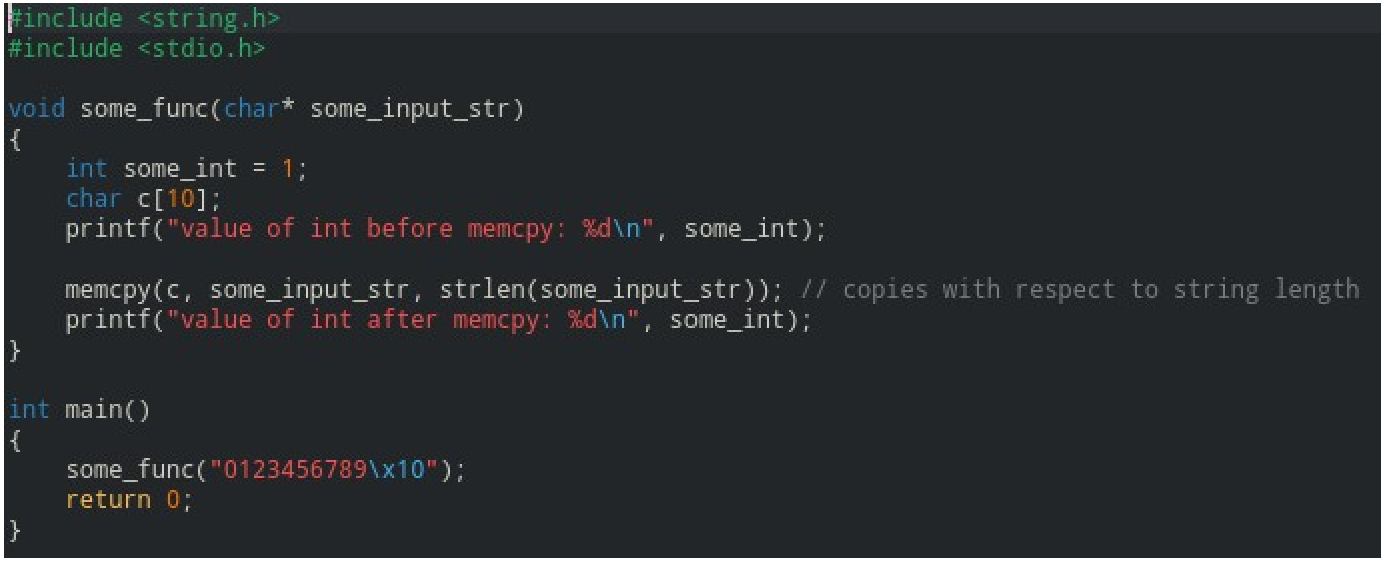
- Modifies the data organisation in the stack frame (or in the heap) to have a canary value
- When destroyed → buffer overflow attack
- Types:
  - Terminator ~: Zero valued canaries, sometimes checks for string terminators
  - Random ~: they are randomly genrated, to prevent attacker from knowing their values
  - Random XOR ~: same as above but also XOR-scrambled with part or all of the control data

Canary



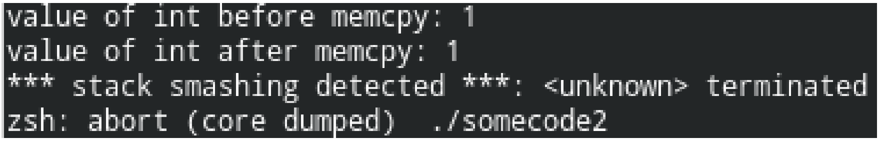
Source: https://ocw.cs.pub.ro/courses/cns/labs/lab-08

Canary - Demonstration



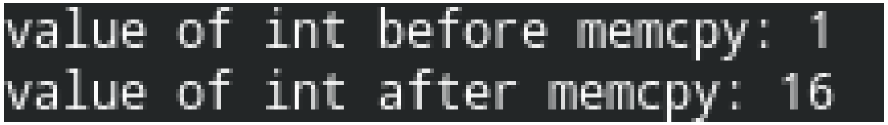
```c
#include <string.h>
#include <stdio.h>

void some_func(char* some_input_str)
{
    int some_int = 1;
    char c[10];
    printf("value of int before memcpy: %d\n", some_int);

    memcpy(c, some_input_str, strlen(some_input_str)); // copies with respect to string length
    printf("value of int after memcpy: %d\n", some_int);
}

int main()
{
    some_func("0123456789\x10");
    return 0;
}
```

Stack protection is on by default in gcc



```
value of int before memcpy: 1
value of int after memcpy: 1
*** stack smashing detected ***: <unknown> terminated
zsh: abort (core dumped)  ./somecode2
```

Gcc compilation with -fno-stack-protector



```
value of int before memcpy: 1
value of int after memcpy: 16
```

Compiler Based Protection – Pointer Protection

- Generally buffer overflow attacks work by manipulating pointers
- In concept, it is supposed to prevent pointer manipulation
- Approach: XOR-encode pointers before and after they are used
- PointGuard was proposed, but ultimately abandoned

Nintendo 3DS - Ninjhax

https://youtu.be/DOHc5LT-Vr4?t=60

Nintendo Wii – Twilight Hack

https://youtu.be/zaRhyEUOk44?t=75