

HackerFrogs Afterschool

Binary Hacking Basics: Part 1

Class:

Binary Hacking

Workshop Number:

AS-BIN-01

Document Version:

1.0

Special Requirements:

None

Name?

AAAAA

AHACK

Welcome to HackerFrogs Afterschool!

HackerFrogs Afterschool is a cybersecurity program for learning beginner cybersecurity skills across a wide variety of subjects.

This workshop is the intro class to binary hacking



What is Binary Hacking?

Binary Hacking is the interaction with a compiled computer program (a.k.a binary executable) which results in an unexpected response that is beneficial to the user interacting with the program



What is Binary Hacking?

In this introductory workshop, we'll be taking a look at a couple of examples of compiled programs that have been coded in a such a way that they can be hacked by anyone who can identify the vulnerability.



What are CTFs?

HackerFrogs
Afterschool classes
prefer to incorporate
CTF games into
classes for a more
interactive
experience, but
what are CTFs?



What are CTFs?

Cybersecurity Capture The Flag (CTF) games are training exercises where the goal of the exercise is to “capture the flag” through use of cybersecurity skills.



What are CTFs?

In this context,
“capture” means to
gain access to a file
or other resource,
and “flag” refers to
a secret phrase
or password.



Pico CTF

The CTF game we will be playing to learn basic binary hacking is called Pico CTF, which is one of the most well-known and well-respected CTF games, and is affiliated with Carnegie Mellon University.



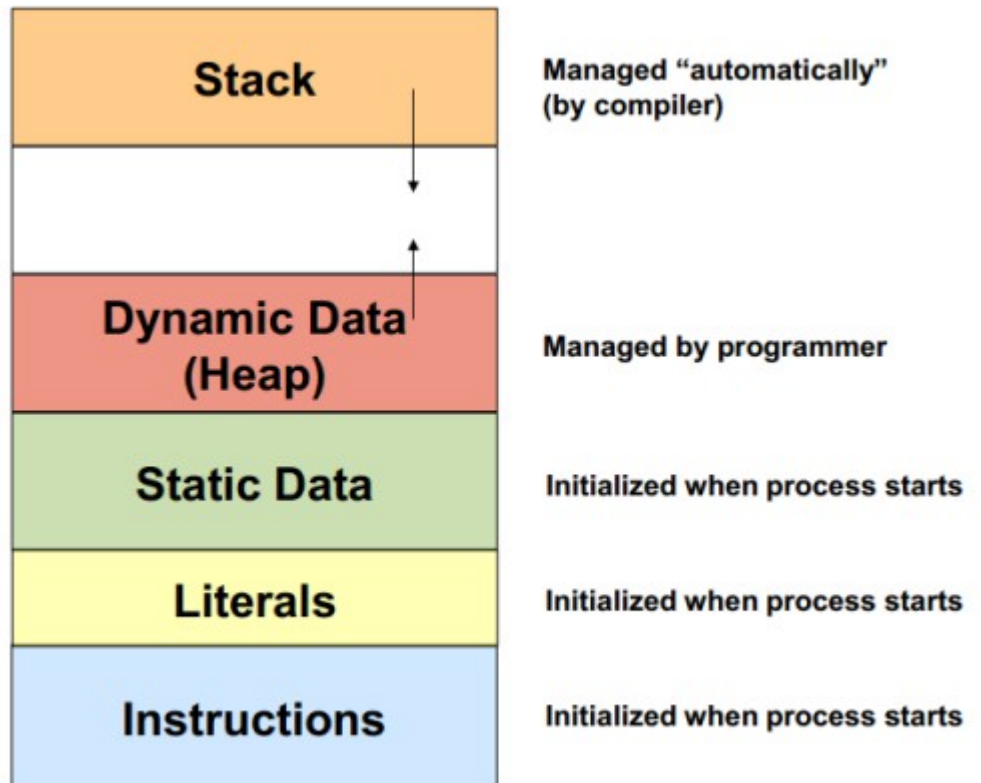
Pico CTF

The Pico CTF game is made up of many challenges across different categories and difficulty levels. For this lesson, we'll be looking at a couple of Pico CTF challenges in the binary hacking category.



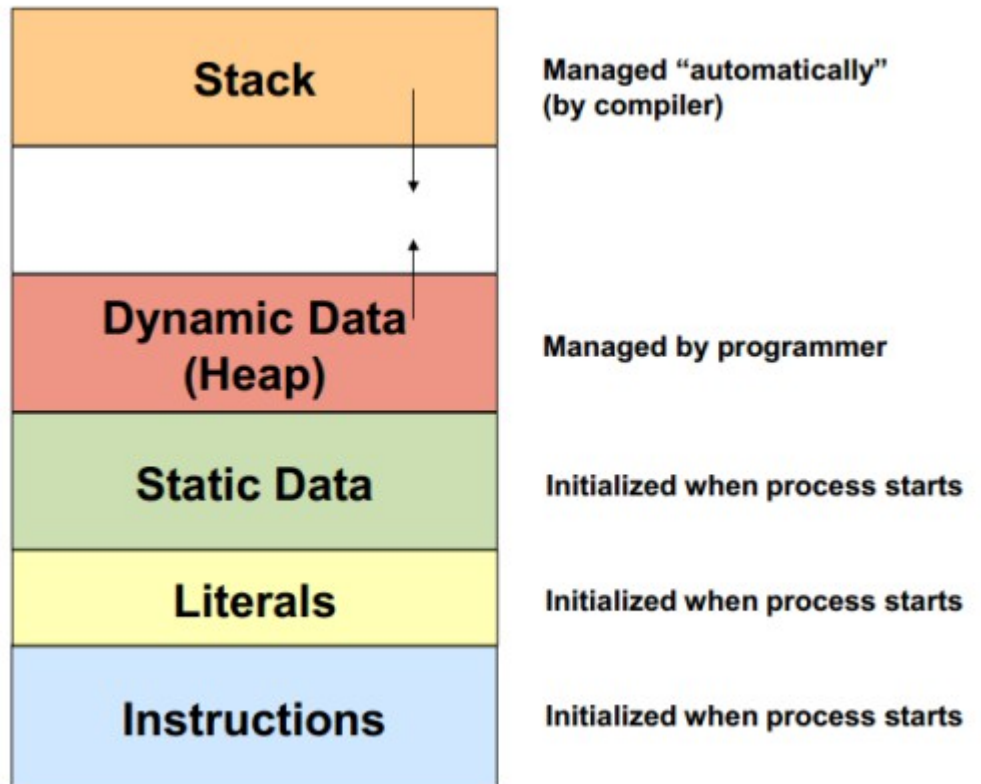
Intro to Memory Hacks

A well-known type of binary hacks involve memory vulnerabilities, which occur when a program is written in an insecure way--



Intro to Memory Exploits

which allows users to inject arbitrary data into a program's memory space, which can result in any or all of the following:



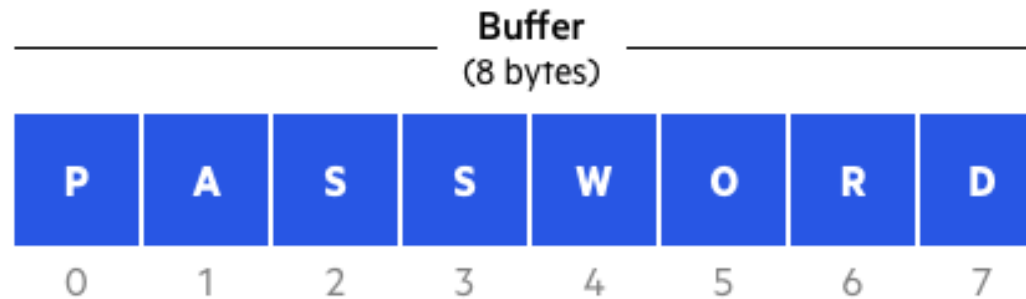
Intro to Memory Hacks

Crash the program

Allow access to other functions in the program

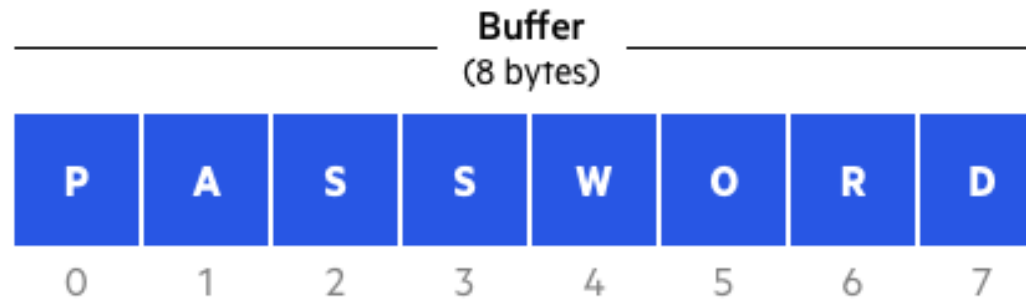
Allow arbitrary command execution on the
program's host system

What are Memory Buffers?



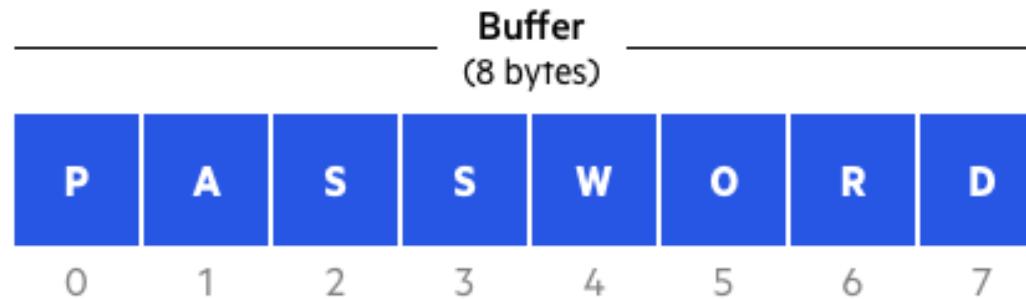
In order to understand memory hacks, we must first understand the concept of memory buffers.

What are Memory Buffers?



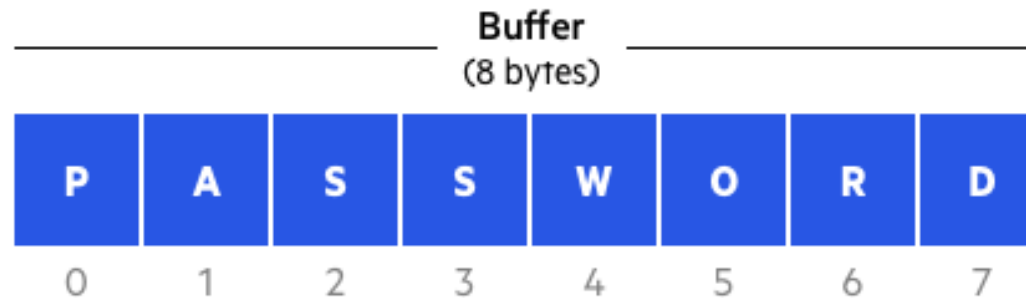
In programming, memory buffers are sections of a program's memory space that are set aside to hold data, e.g., user input.

What are Memory Buffers?



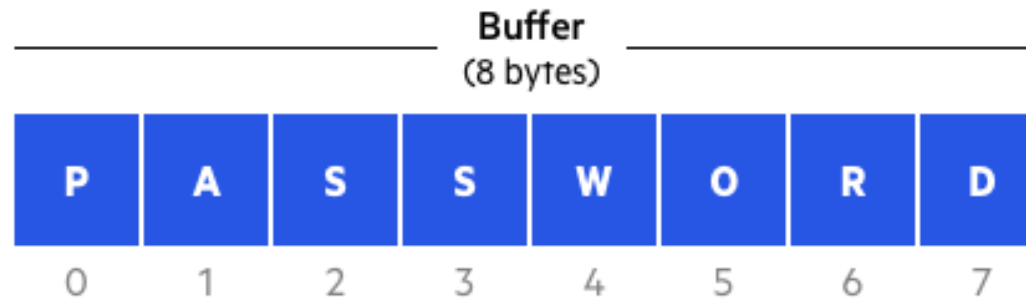
In C programming, the amount of space allocated to each memory buffer must be specified.

What are Memory Buffers?



E.g., a buffer meant to store a user's password may be set to 8 bytes (each character in a password requires 1 byte), so the maximum length of that password should be 8 characters.

What are Memory Buffers?



But what happens if the program receives input for that password which exceeds 8 characters?

What are Memory Buffers?



Data in excess of 8 bytes will overflow out of the specified memory buffer into the neighboring memory space, overwriting the data that was there previously.

What are Memory Buffers?



In programming, the act of data escaping the memory buffer allocated to it is known as buffer overflow.

Sigsegv Event Handling /w Pico Buffer Overflow 0

To learn more about buffer overflows, let's look at
a challenge in Pico CTF:

<https://play.picoctf.org/practice/challenge/257>

What are Segmentation Violations?

Segmentation violations, aka segmentation faults are a condition that causes programs to crash. They occur when a program attempts to read or write to an illegal memory space.




What are Segmentation Violations?

In this context, an illegal memory space is a memory location that doesn't exist. In the context of buffer overflows hacks, they often cause segmentation violations, which lead to program crashes



What is the Gets Function?


```
printf("Input: ");  
fflush(stdout);  
char buf1[100];  
gets(buf1);  
vuln(buf1);  
printf("The program will exit now\n");  
return 0;
```



The **gets** function is used to save user input to a variable in C programming

What is the Gets Function?


```
printf("Input: ");  
fflush(stdout);  
char buf1[100];  
gets(buf1);  
vuln(buf1);  
printf("The program will exit now\n");  
return 0;
```



However, the **gets** function doesn't check the size of the user input, which means it can be used for buffer overflow attacks

What is the Gets Function?

```
printf("Input: ");  
fflush(stdout);  
char buf1[100];  
gets(buf1);  
vuln(buf1);  
printf("The program will exit now\n");  
return 0;
```



If the user inputs a string that is longer than 100 characters, a buffer overflow will occur

What is the Strcpy Function?

```
void vuln(char *input){  
    char buf2[16];  
    strcpy(buf2, input);  
}
```

The program also makes use of the **strcpy** function, which copies the value of one variable to another

What is the Strcpy Function?

```
void vuln(char *input){  
    char buf2[16];  
    strcpy(buf2, input);  
}
```

Here, it copies the value of the **input** variable to the **buf2** variable

What is the Strcpy Function?

```
void vuln(char *input){  
    char buf2[16];  
    strcpy(buf2, input);  
}
```

Strcpy is another insecure function, because it does not check the size of the data that is being moved

What is the Strcpy Function?

```
void vuln(char *input){  
    char buf2[16];  
    strcpy(buf2, input);  
}
```

Since the size of the memory buffer for the **buf2** variable is 16 bytes, if the size of the data in the **input** variables is larger than 16 bytes, buffer overflow will occur

Sigsegv Event Handling /w Pico Format String 0

To learn more about buffer overflows, let's look at another challenge in Pico CTF:

<https://play.picoctf.org/practice/challenge/433>

What is the Scanf Function?

```
char choice1[BUFSIZE];  
scanf("%s", choice1);
```

```
#define BUFSIZE 32
```

The **scanf** function is used to take in formatted user input and save it to a variable

What is the Scanf Function?

```
char choice1[BUFSIZE];  
scanf("%s", choice1);
```

```
#define BUFSIZE 32
```

Like the other functions discussed today, scanf does not check the size of user input by default, so it can be used for buffer overflow hacks

What is the Scanf Function?

```
char choice1[BUFSIZE];  
scanf("%s", choice1);
```

```
#define BUFSIZE 32
```

In this program, **BUFSIZE** is set to 32, and it is the allotted buffer size for the **choice1** variable

What is the Scanf Function?

```
char choice1[BUFSIZE];  
scanf("%s", choice1);
```

```
#define BUFSIZE 32
```

Here, if the user inputs more than 32 bytes to the program, then a buffer overflow occurs

What is Memory Address Space?

In order to study memory exploitation further, we have to familiarize ourselves with program memory address space.

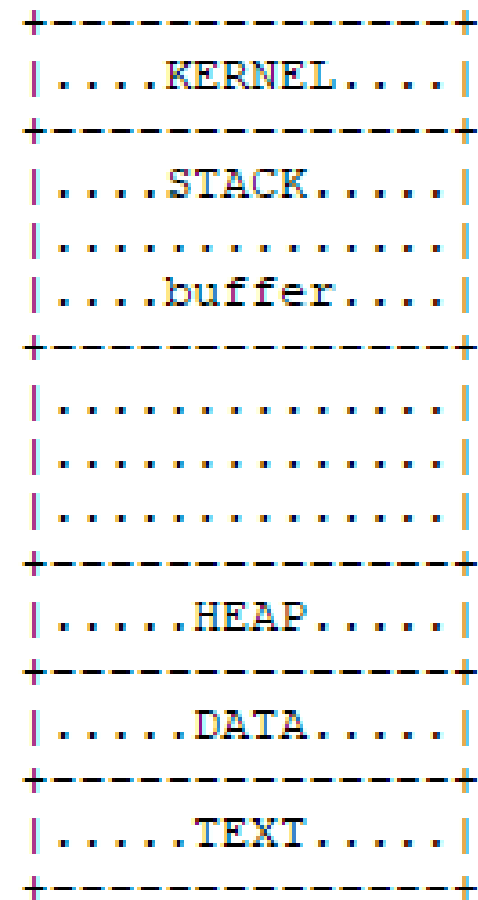
Program Memory Address Space

```
+-----+
| . . . . KERNEL . . . . |
+-----+
| . . . . STACK . . . . |
| . . . . . . . . . . |
| . . . . buffer . . . . |
+-----+
| . . . . . . . . . . |
| . . . . . . . . . . |
| . . . . . . . . . . |
+-----+
| . . . . . HEAP . . . . |
+-----+
| . . . . . DATA . . . . |
+-----+
| . . . . . TEXT . . . . |
+-----+
```

What is Memory Address Space?

When a program launches, a portion of the computer's memory is allocated to it, forming the program's memory space. The data in the memory space is divided into chunks of 4 or 8 bytes, for 32-bit and 64-bit programs, respectively.

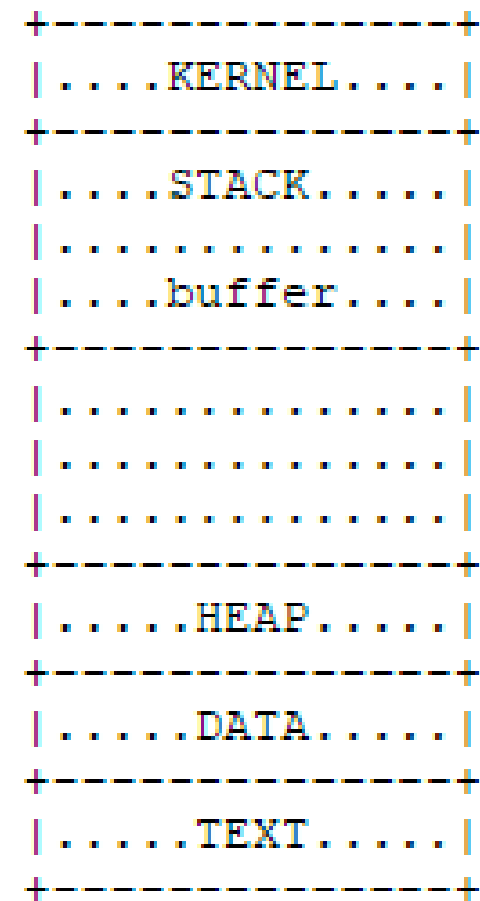
Program Memory Address Space



What is Memory Address Space?

In a 32-bit program, the lowest memory address is 00000000 (4 bytes, two characters per byte), or 0x00000000, and the highest address is FFFFFFFF or 0xFFFFFFFF

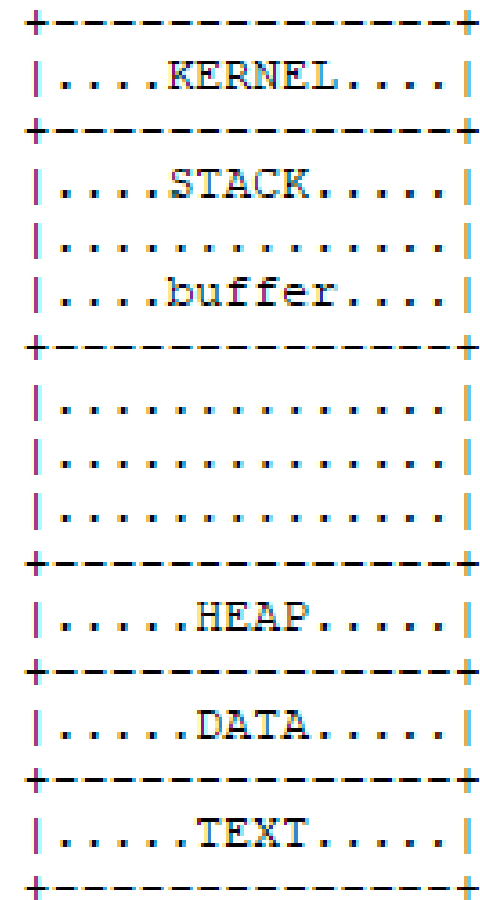
Program Memory Address Space



What is Memory Address Space?

The 0x pre-pended to the address indicates hexadecimal notation. There are five major divisions in the program memory space: the kernel, stack, heap, data, and text.

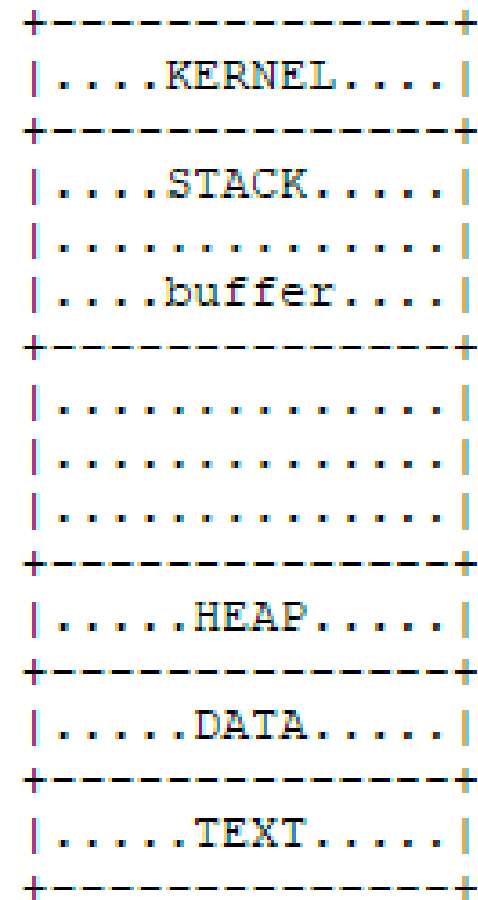
Program Memory Address Space



What is Kernel Memory?

The **kernel** section occupies the lowest memory addresses and contains data related to the computer's operating system processes.

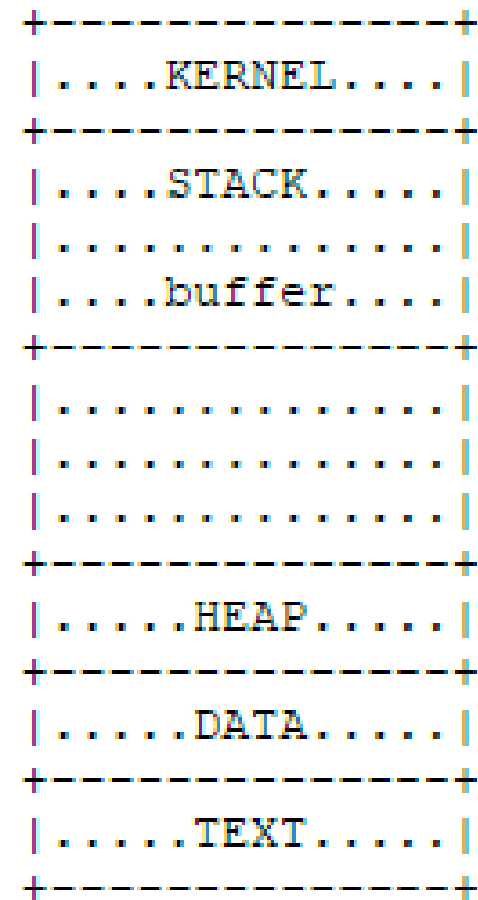
Program Memory Address Space



What is Stack Memory?

The **stack** section follows the kernel section, and it contains variables that store program parameters. It is a prime candidate for buffer overflow hacks

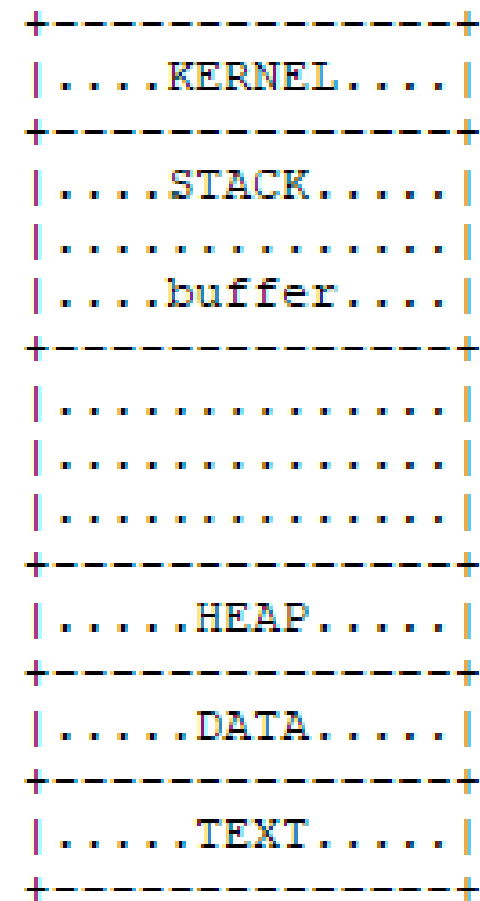
Program Memory Address Space



What is Stack Memory?

As data is added to the stack, it progresses to higher memory addresses, but any buffers overflows that occurs on the stack progress towards lower memory addresses.

Program Memory Address Space



Sigsegv Event Handling /w Pico Format String 0

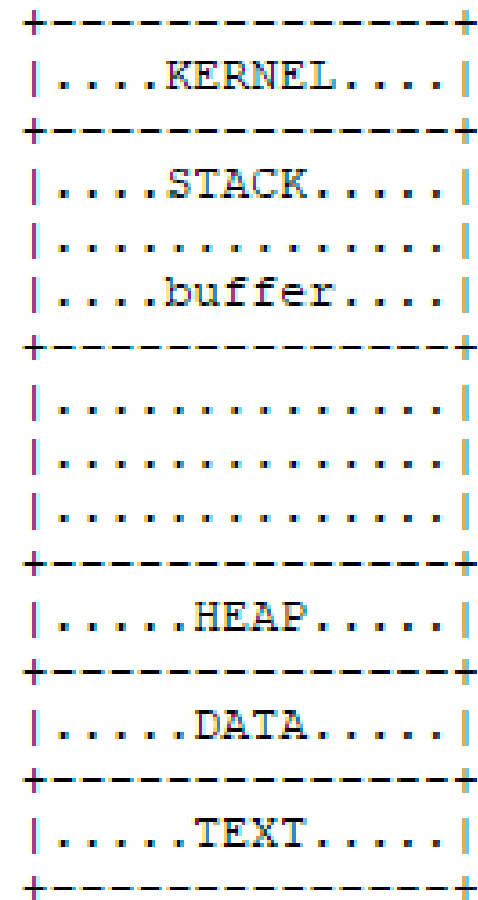
To learn more about buffer overflows, let's look at another challenge in Pico CTF:

<https://play.picoctf.org/practice/challenge/438>

What is Heap Memory?

Heap memory is the section of program memory which holds variables that are intentionally allocated through C functions like **malloc**

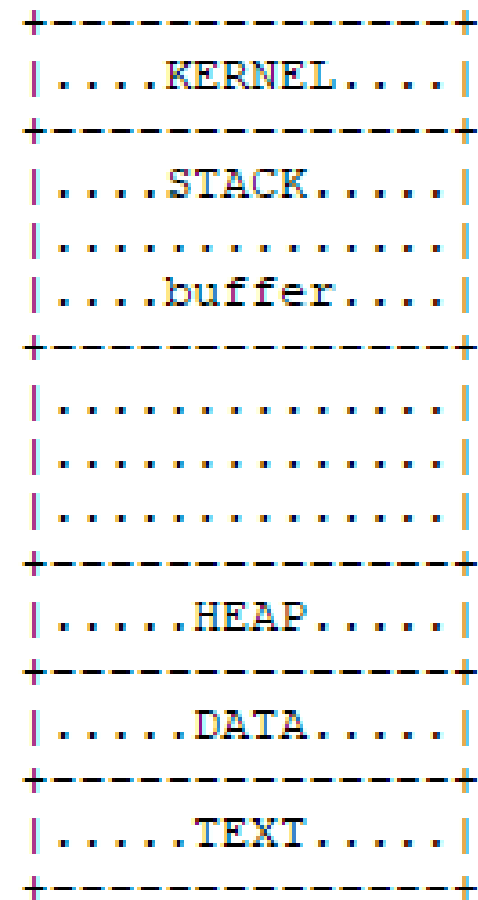
Program Memory Address Space



What is Heap Memory?

Buffer overflows that happen in heap memory are different from stack overflows in that they may be used to overwrite the values of other program variables

Program Memory Address Space



What is the Scanf Function?

```
void write_buffer() {  
    printf("Data for buffer: ");  
    fflush(stdout);  
    scanf("%s", input_data);  
}
```

```
#define INPUT_DATA_SIZE 5
```

Here, if the program takes input using the **scanf** function, and 5 bytes are allocated to the memory buffer

Scanf Function Vulnerability

+-----+-----+	
[*] Address	→ Heap Data
+-----+-----+	
[*] 0x5634f920f6b0	→ pico
+-----+-----+	
[*] 0x5634f920f6d0	→ bico
+-----+-----+	

And the program lets you see the memory addresses of the **input_data** and **safe_var** variables, respectively

Scanf Function Vulnerability

+-----+-----+	
[*] Address	→ Heap Data
+-----+-----+	
[*] 0x5634f920f6b0	→ pico
+-----+-----+	
[*] 0x5634f920f6d0	→ bico
+-----+-----+	

Since the address of **input_data** is lower than **safe_var**, and they exist on the heap, we can overflow **input_data** to overwrite **safe_var**

Summary



Let's review the concepts we learned in today's workshop:

Until Next Time, HackerFrogs!

