430/530 – Buffer Overflow Lab

Name: Caleb Lemmons

ASLR

Cite any sources used to assist you with your answers.

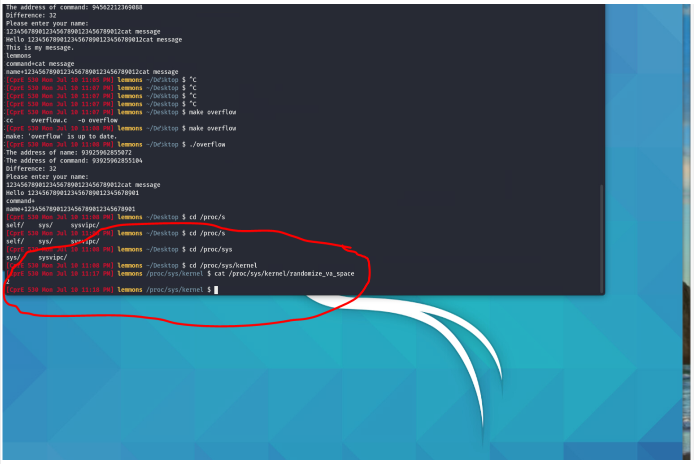
Briefly describe what ASLR is/does and what purpose it serves.

Address Space Layout Randomization is a technique that aids in preventing the exploitation of memory corruption vulnerabilities. ASLR does this by randomizing the memory layout of a running program making it harder for attackers to find and exploit vulnerabilities, since they have to guess the right positions of the files or programs they are trying to exploit.

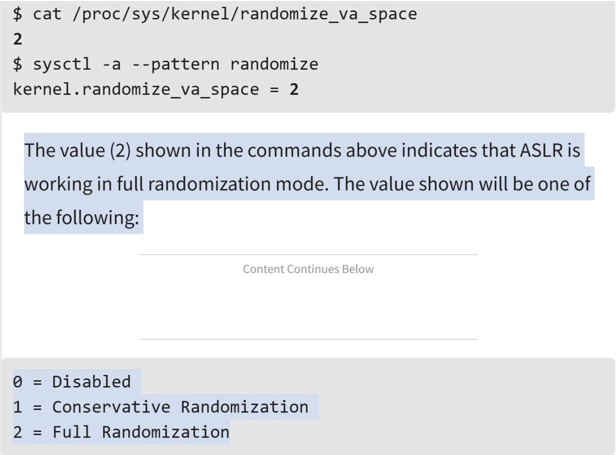
Determine if your Kali VM in ISELab is using ASLR. Justify your answer, i.e., explain how you know ASLR is being used on your Kali VM. It may be helpful to provide a screen capture to supplement your answer.

Yes, ASLR is being used in Full Randomization mode on this VM. With the help of some research I found a way to search for the chain of commands used to check the state of he ASLR on this machine. The NetworkWorld article explained the process in detail from which directories to be in to what code to input to get the desired output. Below is an excerpt of the article that was important to answering this question along with the code I used to find the output.

https://www.networkworld.com/article/3331199/what-does-aslr-do-for-linux.html



```
The address of command: 94562212369088
Difference: 32
Please enter your name:
123456789012345678901234567890l2cat message
Hello 123456789012345678901234567890l2cat message
This is my message.
lemmons
command+cat message
name+123456789012345678901234567890l2cat message
[CprE 530 Mon Jul 10 11:05 PM] lemmons ~/Desktop $ ^C
[CprE 530 Mon Jul 10 11:07 PM] lemmons ~/Desktop $ ^C
[CprE 530 Mon Jul 10 11:07 PM] lemmons ~/Desktop $ ^C
[CprE 530 Mon Jul 10 11:07 PM] lemmons ~/Desktop $ ^C
[CprE 530 Mon Jul 10 11:07 PM] lemmons ~/Desktop $ make overflow
cc     overflow.c   -o overflow
[CprE 530 Mon Jul 10 11:08 PM] lemmons ~/Desktop $ make overflow
make: 'overflow' is up to date.
[CprE 530 Mon Jul 10 11:08 PM] lemmons ~/Desktop $ ./overflow
The address of name: 93925962855072
The address of command: 93925962855104
Difference: 32
Please enter your name:
123456789012345678901234567890l2cat message
Hello 1234567890123456789012345678901
command+
name+1234567890123456789012345678901
[CprE 530 Mon Jul 10 11:08 PM] lemmons ~/Desktop $ cd /proc/s
self/    sys/     sysvipc/
[CprE 530 Mon Jul 10 11:08 PM] lemmons ~/Desktop $ cd /proc/s
self/    sys/     sysvipc/
[CprE 530 Mon Jul 10 11:08 PM] lemmons ~/Desktop $ cd /proc/sys
sys/     sysvipc/
[CprE 530 Mon Jul 10 11:08 PM] lemmons ~/Desktop $ cd /proc/sys/kernel
[CprE 530 Mon Jul 10 11:17 PM] lemmons /proc/sys/kernel $ cat /proc/sys/kernel/randomize_va_space
2
[CprE 530 Mon Jul 10 11:18 PM] lemmons /proc/sys/kernel $ 
```



```
$ cat /proc/sys/kernel/randomize_va_space
2
$ sysctl -a --pattern randomize
kernel.randomize_va_space = 2
```

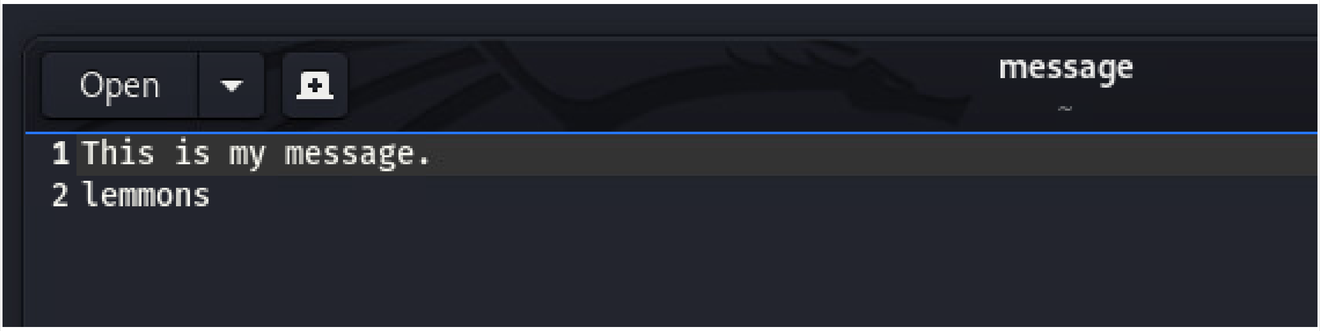
The value (2) shown in the commands above indicates that ASLR is working in full randomization mode. The value shown will be one of the following:

Content Continues Below

```
0 = Disabled
1 = Conservative Randomization
2 = Full Randomization
```

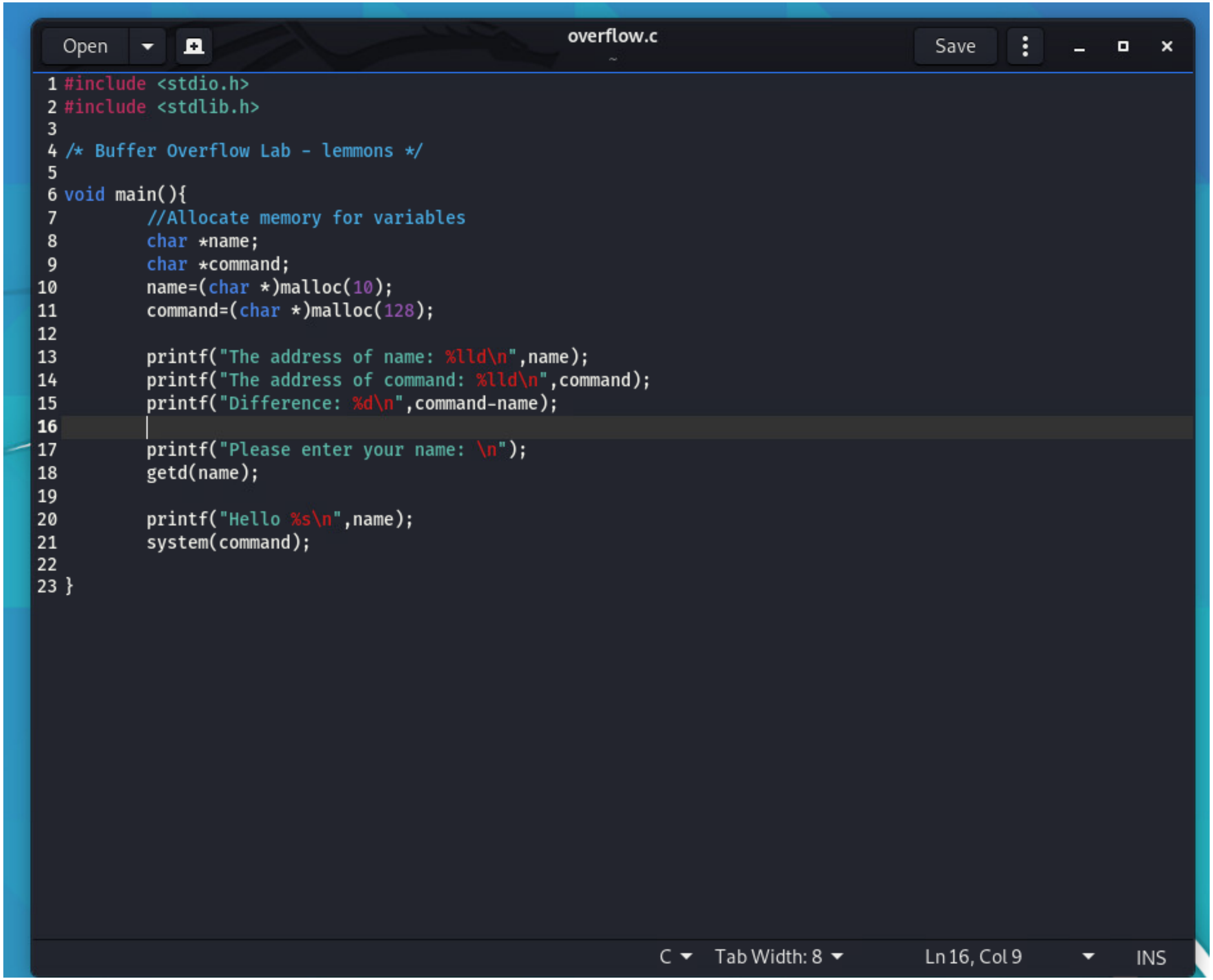
Lab Setup

message file



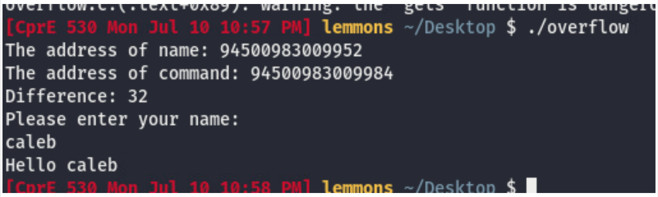
```
This is my message.
lemmons
```

overflow.c code



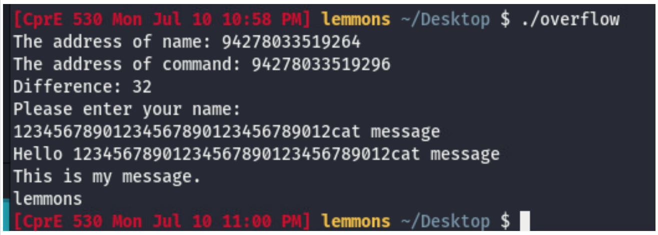
```c
#include <stdio.h>
#include <stdlib.h>

/* Buffer Overflow Lab - lemmons */

void main(){
        //Allocate memory for variables
        char *name;
        char *command;
        name=(char *)malloc(10);
        command=(char *)malloc(128);

        printf("The address of name: %lld\n",name);
        printf("The address of command: %lld\n",command);
        printf("Difference: %d\n",command-name);

        printf("Please enter your name: \n");
        getd(name);

        printf("Hello %s\n",name);
        system(command);

}
```

Experiment 1



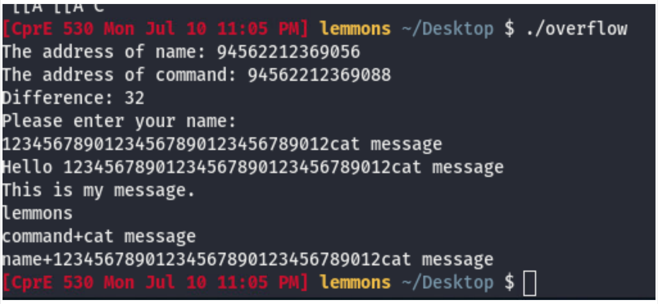
```
overflow.c:(.text+0x89): warning: the `gets' function is dangero
[CprE 530 Mon Jul 10 10:57 PM] lemmons ~/Desktop $ ./overflow
The address of name: 94500983009952
The address of command: 94500983009984
Difference: 32
Please enter your name:
caleb
Hello caleb
[CprE 530 Mon Jul 10 10:58 PM] lemmons ~/Desktop $ 
```

Experiment 2



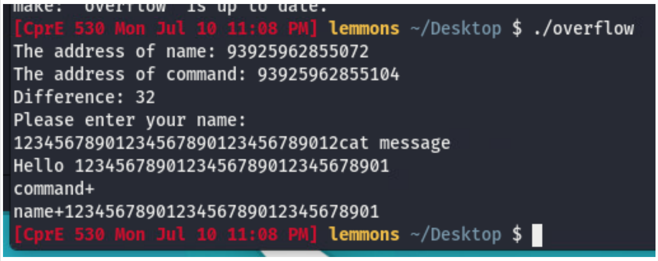
```
[CprE 530 Mon Jul 10 10:58 PM] lemmons ~/Desktop $ ./overflow
The address of name: 94278033519264
The address of command: 94278033519296
Difference: 32
Please enter your name:
12345678901234567890123456789012cat message
Hello 12345678901234567890123456789012cat message
This is my message.
lemmons
[CprE 530 Mon Jul 10 11:00 PM] lemmons ~/Desktop $ 
```

Experiment 3



```
 [[A [[A C
[CprE 530 Mon Jul 10 11:05 PM] lemmons ~/Desktop $ ./overflow
The address of name: 94562212369056
The address of command: 94562212369088
Difference: 32
Please enter your name:
12345678901234567890123456789012cat message
Hello 12345678901234567890123456789012cat message
This is my message.
lemmons
command+cat message
name+12345678901234567890123456789012cat message
[CprE 530 Mon Jul 10 11:05 PM] lemmons ~/Desktop $ 
```

Experiment 4



```
make: 'overflow' is up to date.
[CprE 530 Mon Jul 10 11:08 PM] lemmons ~/Desktop $ ./overflow
The address of name: 93925962855072
The address of command: 93925962855104
Difference: 32
Please enter your name:
12345678901234567890123456789012cat message
Hello 12345678901234567890123456789012345678901
command+
name+12345678901234567890123456789012345678901
[CprE 530 Mon Jul 10 11:08 PM] lemmons ~/Desktop $ 
```

Reminder: If, *at any point* during this lab, you get an error message, such as 'cat: message: No such file or directory,' 'command not found,' or 'sh: 1: <something> not found,' **something is incorrect**, and you will lose points if you do not correct it. You need to debug your program, check your file locations, and check your terminal commands. If you need help diagnosing the problem, contact your TA(s). Please check your images before submitting your PDF.

Lab Reflection

Remember, answer using complete sentences and enough details/specifics to avoid vague answers. Please spend time thinking of a thoughtful response. You may use outside sources to develop your responses but be sure you cite any sources you use.

A. Coding with security in mind and using ASLR are two buffer overflow mitigation strategies discussed in this lab. What are two other ways buffer overflow attacks could be mitigated?

In addition to coding with security in mind and using ASLR, there are a few other ways to mitigate buffer overflow including, Data Execution Prevention (DEP), which is a technique that prevents code from being executed from data pages like stacks or heaps. To add, A technique called Stack Canaries also can help mitigate buffer overflow attacks. Stack Canaries involves placing a small integer, known as a “canary,” before the return pointer of a stack. The canary’s value is checked before the function returns and if it has changed, the program will terminate which would prevent the buffer overflow attack.

B. This lab mentioned the Morris Worm and Heartbleed. Perform Internet research to find another prominent buffer overflow attack. Provide a brief description of the attack and include information such as the attack name, where the buffer vulnerability exists (e.g., specific piece of software, OS, protocol), a CVE number, and how the attack unfolds.

Another prominent buffer overflow attack I came upon was the SQL Slammer worm and it exploited a buffer overflow vulnerability in Microsoft SQL Server 2000 as well as the Desktop Engine (MSDE) 2000. The worm would spread rapidly by generating random IP addresses and sending itself to those addresses, infecting vulnerable systems. It caused widespread denial of service (DoS) attacks and slowed down Internet traffic significantly. The vulnerability is identified by CVE-2002-0649 and is not seen much in modern computing

C. In a paragraph of *at least* five sentences, reflect on what you have accomplished in this lab.

In the Buffer Overflow Lab, I was able to learn about some interesting buffer overflow mitigation strategies and witness how important it is to code with security in mind along. It was interesting to see how one line of code can drastically change the likelihood of a buffer overflow attack being successful or not. I have barely worked with buffer overflows so this lab was a cool interactive introduction.