Project 1 - Part 1: Overflowing the Stack

Goals of the Project:

- Understand and explain how virtual memory is laid out into different regions
- Understand and explain how the stack and heap work
- Understand and explain the basic stack concepts and how it controls program execution
- Understand and explain the concepts of buffer overflow
- Explain how a buffer overflow occurs in detail and what effects it has on the heap

Supplemental Readings:

- *Complete Virtual Memory Systems*: Click Here
- *Memory API*: Click Here
- *Address Spaces*: Click Here

The final deliverable:

The submitted paper must adhere to the following format: (If not, **-10 points**)

- All Diagrams are to be created by the person submitting. No hand drawn diagrams.
  - You are allowed to use any tool to create the diagrams. (i.e draw.io, paint, or excel)
- Written answers to each individual question are limited to a maximum of 200 words
- Font: Times New Roman, Size 11.
- Spacing: Single Spaced with standard 1" margins
- Heading: gt_user_id above page number. (Remove ***gburdell27*** and put your own)
- Submission name format: **`gt_user_id_Project1.pdf`**
  - Example: **`gburdell27_Project1.pdf`**

Information:

Plagiarism:

Plagiarism will not be tolerated! For information: GaTech Academic Honor Code.

- Papers will be run through TurnItIn.com, as well as other plagiarism detectors.
  - If it is deemed that more than 10% of the information contained in the paper was not properly cited, you will be reported to GaTech Office of Student Integrity.
- Papers must be cited in IEEE format.
  Note: Even if you cite a source, you are ***not*** allowed to submit a paper consisting of copy and pasted fragments. You must show understanding and summarize in your own words. (You must cite paraphrasing)

- You **MUST** use the latest version of VirtualBox, downloaded from: VirtualBox
- GDB command cheat sheet: Cheat Sheet

Submission Deadline:

There will be no late projects accepted. The deadline will be set to 11:59pm EST. in canvas. Be sure to turn in your work prior to the deadline in canvas. In prior semesters, students have had issues turning things into canvas at 11:50-11:59pm on the due date. *BE SURE* to turn in your work prior to this time. No late submissions will be accepted. A general piece of advice is to turn in your work a few hours early, continue working on it, and turn it in again nearer the deadline. It is better to get some points for incomplete work than no points for no work. This rule is set in stone. There will be no granted extensions unless they have been granted through the Office of the Dean of Students.

Project Tasks (75 points):

I. Understanding Foundational Concepts - (10 points)

1. What is a pointer?
2. How is memory managed differently between say "C" and "Java"?
3. When is memory allocated and deallocated in "C"? Answer both static and dynamic memory.
4. Would it be smart to use fixed-width integer types? Explain.
5. What is the difference between "compile time" and "run time"?
6. Give an example of a variable declaration and definition at compile time. Then give an example of this variable being redefined during run time. Finally, show an allocation of memory during runtime. Provide code examples in "C".
7. What is the difference between an assembler, compiler, and an interpreter?
8. Is C a compiled language? Is Python? If either answer is no, how is the language processed so code can be run?
9. What is the GNU compiler? How do you invoke the GNU Compiler?
10. What is GDB? In GDB, how would you:
    a. View the processor registers
    b. Set a breakpoint in code
    c. Find the address of an OS Function
    d. Inspect a memory location

II. Understanding Components - (12 points)

Important: Answer the following questions in relation to the 32-bit Linux VM that is being used.

Research each of the following and give an explanation as to how each affects the stack.

1. Address space layout randomization (ASLR)
    - Explain how ASLR can be bypassed without turning it off.
2. Stack Canary
    - Are these vulnerable?
    - If so, how?
3. Stack Buffer Overflow (with no protections enabled)
4. Stack Specific Registers and their purpose (x86 specific for this question)
5. Stack memory allocation on a 32bit OS
6. How do the following lines of code get allocated on the stack? Why are they handled differently by the stack? (minus the obvious 3 byte difference)

```
char buf1[5]; // Line 1
char buf2[8]; // Line 2
```

III. Compiler Flag Options - (12 points)

Research each of the following and give an explanation to how each affects the program/binary.
Important: This is referring to the ***gcc*** *compiler.*

-O0 (This is the letter **O**, then the numeral **0**)
-g
-fno-stack-protector
-z execstack

IV. Buffer and Heap Understanding - (12 points)

All of the following questions refers to the address space of a 32 bit Linux distribution OS.

1. Where in memory is the stack located and in which direction does it grow?
2. How is data stored and organized on the stack?
3. Explain how program control flow is implemented using the stack.
4. Where in memory is the heap located and in which direction does it grow?
5. How is data stored and organized on the heap?
6. What kind of program data goes to the stack versus the heap? Where are global variables stored?

V. Stack Overflow - (29 points)

The VM is located: VM Download Link and the code provided below is in a file **`overflow.c`** which can be downloaded here. Figure out how to compile and run the program. Understand how the overflow works.

Hint: You may need to use the research you did above to get this program to execute correctly.

```
01.  #include <stdio.h>
02.  #include <string.h>
03.
04.  int main (int argc, char *argv [])
05.  {
06.      int accessGranted = 0;
07.      char str1[8];
08.
09.      printf("Please enter a password: ");
10.      scanf("%s", str1);
11.
12.      if (strncmp (str1, "password", 8) == 0)
13.          accessGranted = 1;
14.      if (accessGranted > 0)
15.          printf("Valid input! Access Granted!\n");
16.  }
```

Important: Include a screenshot of your terminal exploiting overflow.c in the write up. If not, **-10 points**.

Stack Overflow Understanding (11 points)

1. What line(s) in the above code are vulnerable and why?
2. Could a user receive the “Valid input! Access Granted!” output without inputting “password” as their password? If so, how?
3. How could you change the code to make this exploit not possible?
4. Would a similar program have the same vulnerability in a strongly typed language? Why or why not?

Stack Overflow Diagram (18 points)

Create a diagram that shows what the stack would look like before and after the given input (or create a single diagram that would show both). This diagram has to be 100% created by you. You are ***not*** allowed to copy/paste anything from anywhere else. Your diagram must include:

- The order of parameters (if applicable), return address, saved registers (if applicable), and local variable(s).
- The sizes in bytes.
- The overflow direction in the stack.
- Size of the overflowing buffer.

Important: The sample input entered that should be shown in your diagram is “OverflowDone”.