Unit 1: The Host Section

Lesson 1: Building Blocks for Computing 1: Digital Logic and Review of C Programming

a) Identify the logic circuit gates and reproduce the truth tables for NOT, AND, NAND, OR, and NOR gates.
b) Given a schematic of a logic circuit, determine the corresponding Boolean expression for the circuit output.
c) Define the term “cyber-physical system (CPS)” and provide two examples.
d) Demonstrate the ability to write simple C programs that perform keyboard input, screen output and simple arithmetic.
e) Discuss the role of the operating system in bridging the gap between hardware and user applications and services.
f) Demonstrate the ability to create, edit, compile and execute C programs in a Linux environment.
g) Demonstrate the ability to analyze C programs that employ if-else statements and for loops.

Lesson 2: Building Blocks for Computing 2: Transistors and Review of Numbering

a) Define the term transistor and give a brief description of its construction and operation.
b) Identify two different types of transistors by their symbology.
c) List the precautions to be taken when working with transistors and describe ways to test them.
d) Explain how the transistor can be used in digital logic.
e) Explain the purpose and function of various hardware components found in a computer system: CPU, Hard drive, Ram.
f) Review digital data and its binary representation.
g) Convert between binary and decimal notation.
h) Convert between hexadecimal and decimal notation.
i) Convert between hexadecimal and binary notation.
j) Evaluate how characters are stored using ASCII notation.

Lesson 3: Arrays and More C Functionality

a) Describe how an array is stored in memory.
b) Define a string, and describe how strings are stored.
c) Describe the implications of reading or writing beyond the boundary of an array.
d) Describe how to change the values of individual array elements.
e) Demonstrate the ability to analyze C programs that employ if-else statements and for loops.
f) Apply Boolean logic to evaluate how selection structures can alter program flow.
Lesson 4: Main Memory Mechanics

a) Describe the organization and contents of main memory when a program is being executed.
b) Demonstrate the ability to analyze a C program and identify the corresponding assembly language instructions generated.
c) Explain and demonstrate how data is stored in memory for integers, floats and addresses (e.g. little endian).

Lesson 5: The Debugger

a) Demonstrate the ability to debug a running C program in memory, to include the inspection of processor registers and arbitrary memory locations
b) Analyze existing programs for flaws with the gdb debugger.

Lesson 6: Intro to Pointers

a) Explain the operation of the address operator.
b) Describe the relationships that exist between pointers, arrays and strings.
c) Differentiate between the value of a pointer and the address of a pointer.

Lesson 7: User Defined Functions and Stack Mechanics

a) Demonstrate the ability to analyze simple programs that use library and user defined functions.
b) Describe the organization and contents of a program’s stack throughout a program’s execution.
c) Demonstrate the ability to examine the stack values of a running program using the debugger.

Lesson 8: Buffer Overflow Intro

a) Describe the buffer overflow attack, determine what features of C make it possible, and identify who is responsible for memory management in C.
b) Demonstrate the ability to craft simple buffer overflow attacks.
c) Explain how specific buffer overflow attacks work by describing stack operations.

Lesson 9: Privilege Management

a) Analyze programs that submit input via the command line.
b) Describe how permissions are managed and controlled in a multi-user OS environment.
c) Explain how users can be afforded the limited ability to execute commands with escalated privileges.
Lesson 10: A Real Buffer Overflow

a) Describe how a buffer overflow attack can be used to gain root access to a computer.
b) Describe two techniques that a hacker can use to make it simpler to craft a buffer overflow.
c) Given parameters of an attack and the layout of the stack for a program, determine if the attack could be successful.
d) Describe technical solutions that have been proposed to prevent a program from being exploited by a buffer overflow.