Unit 1: The Host Section

Lesson 0: Prerequisite Knowledge of Number Systems and C programming

a) Explain the purpose and function of various hardware components found in a computer system: CPU, Hard drive, Ram.
b) Convert between binary, hexadecimal and decimal notation.
c) Evaluate how characters are stored using ASCII notation.
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.

Lesson 1: Arrays and Strings

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, including the use of `strcpy()`.
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 2: Main Memory Mechanics

a) Locate variables in a memory diagram and translate to integers or characters as appropriate.
b) Demonstrate the ability to analyze a C program and identify the corresponding assembly language instructions generated.
c) Describe the results of the `mov` assembly language instruction with reference to a memory diagram.
d) Explain and demonstrate how data is stored in memory for integers and addresses (e.g. little endian).
e) Describe the purpose of the registers `eip`, `ebp`, and `esp`.

Lesson 3: 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.
c) Given a memory diagram, state the expected result of particular uses of the examine (x) command.
Lesson 4: 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.
d) Determine the value of a pointer after pointer addition or subtraction operations.

Lesson 5: User Defined Functions and Stack Mechanics

a) Given a C program’s source code, identify all function libraries, function definitions, function calls, function arguments, function parameters, and return statements.
b) Given a C program’s source code, identify main’s variables, function arguments, and function variables.
c) Describe the purpose of the return address and prior ebp.
d) Given a C program’s source code and corresponding assembly language instructions, fill out a diagram of the stack memory at a specified point in program execution.

Lesson 6: 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) Given the source code of a C program vulnerable to buffer overflow, determine the minimum number of characters necessary for the user to enter to overwrite a specified item in memory (such as the return address).

Lesson 7: Privilege Management

a) Given the source code for a C program that accepts command line inputs along with a specified user input, determine the values of argc and argv[0], argv[1], etc.
b) Given the output of a file’s long listing (i.e. from ls –l), determine the identity of the owner as well as the privileges afforded to the owner, the group, and the general public.
c) Describe the effect of the setuid flag on program execution.
d) Differentiate between the use of sudo and setuid.

Lesson 8: Buffer Overflow Attack

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.