Solutions to End of Chapter 10 Problems:

1. Order these three main components of a buffer overflow exploit as they will appear on the stack:

   - **NOP sled**
   - **Shellcode**
   - **Malicious return address**

2. At some point in a C program called `hackthis.c`, a function `donothack` is called. A hacker is attempting to take advantage of a vulnerability in the code to initiate a buffer overflow attack when the function `donothack` is called and a command line argument is copied into memory in the function’s stack frame.

   Analyzing the code and using the debugger on `hackthis.exe`, a sketch of the memory layout emerges as depicted to the right (note: sketch is not to scale). For each of the following values explain why they should or should not be used as the malicious repeated return address in a buffer overflow attack.

   (a) 0x0804834b

   This is not a good malicious return address because it points somewhere in the text segment, not the function’s stack frame where the exploit is.

   (b) 0xbffff700

   This is not a good malicious return address because it points to non-executable data in the stack and would probably crash the program.

   (c) 0xbffff740

   This is a good malicious return address because it is within `donothack`’s stack frame, and is greater than 0xbffff728, where the exploit is loaded.

   (d) 0xbffff7b4

   This is not a good malicious return address because it is not in `donothack`’s stack frame, but instead main’s stack frame.
3. Aside from careful programming and the modification of several specific C commands, list and briefly describe two technical solutions that have been proposed to prevent a program from being exploited by a buffer overflow.

Solution:

Answer should describe two of the following:

a. Stack canaries
b. ASLR
c. Non-executable stack
d. System hardening

4. Explain why the buffer overflow described in this chapter is much more insidious than the buffer overflows described in Chapter 8.

Solution:

The buffer overflow described in Chapter 10 is paired with a privilege escalation through setuid root on the vulnerable program, allowing the hacker to not only execute malicious code, but will full access to the host system. In this case, the hacker gains a root shell, and can then create backdoors, open ports, turn on services, install software, read or modify data, change parameters, etc.. The hacker can take complete control of the system, even lock out legitimate users.