**10.2 List the three distinct types of locations in a process address space that buffer overflow attacks typically target.**

Buffer overflows typically target the stack, the heap, or program data.

**10.4 What are the two key elements that must be identified in order to implement a buffer overflow?**

To implement a buffer overflow, an attacker must both find a buffer to which the attacker can inject data and must know how the program uses memory around the buffer.

**10.6 Describe how a stack buffer overflow attack is implemented.**

A stack buffer overflow typically involves injecting data using a buffer overflow that overwrites the return address of the current stack frame to a chosen place in memory that was also overwritten as part of the attack and executing code at the new return address.

**10.8 What restrictions are often found in shellcode, and how can they be avoided?**

Shellcode must be position independent, meaning it should work no matter where in memory it is loaded, and it must contain no null values. To circumvent the no nulls rule, code can generate null values at runtime by XORing a register with itself. To make the code position independent, a NOP sled can be used to provide a wide landing zone for the injected return address to lead to, and it can use the call instruction to find its location in memory.

**10.10 List some of the different operations an attacker may design shellcode to perform.**

Attackers with shellcode might disable firewalls, create remote shells that they can login to, or break out of restricted execution environments.

**10.12 List and briefly describe some of the defenses against buffer overflows that can be used when compiling new programs.**

To protect against buffer overflows a programmer should select a language that protects against buffer overflows, use safe coding practices such as bounds checking, use safe libraries, and could include canaries in the call stack to detect changes in the return address.

**10.14 Describe how a return-to-system-call attack is implemented and why it is used.**

A return-to-system-call attack overwrites the stack return address to point to a known system call, so that no executable code is injected. This attack is used because some systems have a non executable stack to prevent buffer overflow attacks.

**10.16 Describe how a global data area overflow attack is implemented.**

A global data area overflow attack attempts to overwrite function pointers, which are stored in global data, so that the attacker may inject code and force the program to execute it.