## Chapter 10: Buffer Overflow Attacks and Others

A buffer overflow, also known as a buffer overrun or buffer overwrite, is defined in the NIST *Glossary of Key Information Security Terms* as: "A condition at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system."

1. **C Commands**
   a. **Unsafe**
      i. char *strcpy (char *dest,  const char *src)
      ii. char *strcat (char *dest, const char *src)
      iii. char *gets (char *s)
      iv. size_t strlen(const char *s);
      v. int scanf ( const char *format, … )
   b. **Safe**
      i. libc versions  strncpy(), strncat()  are misleading
         1. strncpy()   may leave string unterminated.
         2. API warns you not to use strcat or strncat

```
SECURITY CONSIDERATIONS
     The strcat() function is easily misused in a manner which enables mali-
     cious users to arbitrarily change a running program's functionality
     through a buffer overflow attack.  (See the FSA.)

     Avoid using strcat().  Instead, use strncat() or strlcat() and ensure
     that no more characters are copied to the destination buffer than it can
     hold.

     Note that strncat() can also be problematic.  It may be a security con-
     cern for a string to be truncated at all.  Since the truncated string
     will not be as long as the original, it may refer to a completely differ-
     ent resource and usage of the truncated resource could result in very
     incorrect behavior.  Example:
```

2. **Heap Spraying**
   a. Technique used in exploits to facilitate arbitrary code execution.
   b. Write a certain sequence of bytes at a predetermined memory location, then exploit that to facilitate the execution of arbitrary malicious code.
   c. Easily done with JavaScript

```
<SCRIPT language="text/javascript">
 shellcode = unescape("%u4343%u4343%...");    // allocate in heap
 overflow-string = unescape("%u2332%u4276%...");

 cause-overflow(overflow-string );       // overflow  buf[ ]
</SCRIPT>
```

   d. Pointing a function pointer (unescaped) almost anywhere in heap will cause shellcode to execute.

3. **Integer Overflows**
    a. When an arithmetic operation creates a numeric value outside the range for the number
        i. Higher than the maximum
        ii. Lower than the minimum
    b. Integer overflows are listed as the 8th most dangerous software error
        i. CWE 2019
        ii. Lead to buffer overflows
    c. C strlen – why?

4. **Double free**
    a. Freeing a resource more than once can lead to memory leaks.
    b. The allocating data structure gets corrupted and can be exploited by an attacker.

```
a = malloc(10);     // 0xa04010
b = malloc(10);     // 0xa04030
c = malloc(10);     // 0xa04050

free(a);
free(b);  // To bypass "double free or corruption (fasttop)" check
free(a);  // Double Free !!

d = malloc(10);     // 0xa04010
e = malloc(10);     // 0xa04030
f = malloc(10);     // 0xa04010   - Same as 'd' !
```

    c. Now, 'd' and 'f' pointers point to the same memory address.
    d. Any changes in one will affect the other

5. **Format String Vulnerabilities**
    a. Format String Vulnerabilities are considered a channeling problem
    b. Happens if two different types of information channels are merged into one
    c. Special escape characters are used to distinguish which channel is active
    d. C strcpy or strncpy or printf

6. **Defenses**
    a. Rewrite software in a type safe language (Java, Go, Rust)
    b. Platform defenses: prevent attack code execution
    c. Add runtime code to detect overflows exploits
        i. StackGuard,
            1. Run time tests for stack integrity
        ii. Control Flow Integrity
            1. Restricts the control-flow of an application to valid execution traces
        iii. LibSafe
            1. The libsafe library protects a process against the exploitation of buffer overflow vulnerabilities
            2. Works with any existing pre-compiled executable
    d. Mark stack and heap as non-executable