Assignment 1

1. While the *safe_sqrt* function appears to be safe at first glance, rigorous execution of the program shows that the NaN check is not being performed properly. When passing in a value of -1, the value of NaN is returned from the function without triggering the conditional statement. While this will not result in vulnerabilities such as escalating of privilege or code execution, it can be used to corrupt other pieces of system data. If an attacker were to use the *safe_sqrt* function in a particular manner, other values in the system could become NaN. A NaN value could be detrimental to any type of business or money handler.

2. Upon first inspection, the *snprintf* function appears to be a likely place for a vulnerability. In fact, there is a possible vulnerability within *snprintf* because the rest of the function *CD_KEY_CHECKER* is working in wide characters (*wchar*). Wide characters are twice the size of regular characters and therefore require specialized functions to handle the data type. Instead of using *snprintf*, it is advisable to use *swprintf* which is designed for wide characters. Improper use of *snprintf* could result in buffer problems and memory access issues. Another issue is the conditional use of the *free* function. It is good that the *cd_key* pointer is being set to *NULL* after having been *free*'d, but the function does not guarantee that *cd_key* will be *free*'d. In the event that the key matches the master key, the function will return without ever calling *free*. If this is unintentional, the remaining *cd_key* pointer could grant access to memory locations unwontedly.

3. An important fact for all C programmers to remember is that using safe functions does not make all code safe. A set of safe functions can be used unsafely. This is a clear example of safe functions being used unsafely. Because a string is terminated using the null terminator, it is vitally important that all strings contain a null terminator. If the string does not contain a null terminator, reading the string for a sufficient length can result in reading memory that is not actually contained in the string. The function *strncpy* does not include a null terminator when copying unless the length of the string is smaller than the destination array. For example, if the string length is 5 and the array length if 6, the last element in the array will be a null terminator. If the string length is 6 and the array length is 6, no null terminator will be included. Therefore, when using other functions like *strcmp* that rely on the null terminator, the function will read into memory beyond the end of the string. If an attacker is clever, he could use this vulnerability to determine the contents of subsequent memory addresses relative to the string. If the addresses are, for example, function pointers, malicious code could be forced to execute by overwriting those memory addresses.

(http://blogs.msdn.com/b/oldnewthing/archive/2005/01/07/348437.aspx)

4. There is a vulnerability that exists between two functions and is related in both cases to format string specifiers. There are a number of format string specifiers in the C language – some of which can be extremely dangerous if used in a specific manner. In this example, both *syslog* and *fprintf* functions have not been given format specifiers; instead, the user is capable of arbitrarily providing a set of specifiers. Most prominently, the *%n* specifier can be used to overwrite arbitrary memory addresses. In both functions, *syslog* and *fprintf*, there should be an additional argument specifying exactly which format specifiers to use. For example, *syslog(SYSLOG_WARNING, %s, buf)* which prevents the user from specifying arbitrary format specifiers. Source: "Sockets, Shellcode, Porting, & Coding" by James C. Foster.

5. Firstly, the *spectrum_analysis* function accepts an integer parameter *len* which is later used blindly in a *malloc* function call to create the *temp_spec* pointer. If a malicious user were to pass an invalid value as *len*, the *malloc* function will attempt to allocate the requested memory. However, if *malloc* fails (e.g. excessively large memory request or negative value), the function will return a *NULL Pointer*. Because there are no checks to see if *temp_spec* is *NULL*, when *temp_spec* is accessed later in the function, the system will crash because it cannot access a *NULL Pointer*. Secondly, because the threads share the same heap there is a potential for an attacker to corrupt data values across multiple threads. The variable *max_log_norm* is declared globally and is therefore shared across all threads of execution. If an attacker were clever, he could tamper with the value of *max_log_norm* which would then affect calculations from other threads. Given that this system is used to detect toxin levels in a water sample, incorrect values could raise major alarms and cause serious problems. An attacker having potential to corrupt these reported values is a major security concern.