Sample Midterm Answers

1. Here is a fragment of code from a program that reads data from a file into a dynamically allocated part of memory. There are at least 3 things in this code that make it very non-robust. Find any 3, say why each is a (potential) problem, and how you would fix each. (This question asks about robustness, not commenting style – the comments are just there to help you figure out what is going on.)

```
/* read nchars characters from the file named filename */
/* and put them into dynamically allocated memory      */
char *load(int nchars, char *filename)
{
      char *p;    /* pointer to allocated memory */
      FILE *fp;   /* pointer to the opened file */

      /* allocate space for nchars char */
      p = malloc(nchars * sizeof(char));
      /* open the file */
      fp = fopen(filename, "r");
      /* read nchars characters from the file that   */
      /* fp points to, and put it in the memory that */
      /* begins at address p                         */
      (void) fread(p, sizeof(char), nchars, fp);
      /* close the file */
      (void) fclose(fp);
      /* return the address of the allocated memory */
      return(p);
}
```

Answer: Here are all the problems I found. See if you can find any others!

a. The return value of *malloc* is not checked for **NULL**. It is subsequently used as an array pointer. This would cause a segmentation violation or bus error. To fix it, change the first line after the declaration to be:

```
/* allocate space for nchars char */
if ((p = malloc((nchars+1) * sizeof(char))== NULL)
      return(NULL);
      /* or other appropriate error handling ... */
```

b. The return value of *fopen* is not checked for **NULL**. It is subsequently used as a file pointer in *getc*. This would cause a segmentation violation or bus error. To fix it, change the line with *fopen* to be:

```
/* open the file */
if ((fp = fopen(filename, "r")) == NULL)
      return(NULL);
      /* or other appropriate error handling ... */
```

c. The argument *nchars* is not sanity-checked. If it is negative, *malloc* will allocate non-positive space; this causes undefined results. Add the following line at the beginning of the function:

```
/* sanity-check argument nchars */
if (nchars < 0)
      return(NULL);
      /* or other appropriate error handling ... */
```

d. The argument *filename* is not sanity-checked. If it is **NULL**, *fopen* will be passed a **NULL** pointer as its first argument, and the result is undefined. Add the following line at the beginning of the function:

```
/* sanity-check argument filename */
if (filename == NULL)
      return(NULL);
      /* or other appropriate error handling ... */
```

e. The loop that reads characters does not check for array overflow. Change the conditional to be:

```
i < nchars && (c = getc(fp)) != EOF
```

2. Why is a precise statement of security requirements critical to the determination of whether a given system is secure?

   Answer: A precise statement of security requirements is critical to the determination of whether a given system is secure because the statement of security requirements underlies the system security policy, which in turn *defines* what "secure" means for that site. Without such a statement, one cannot determine whether the system is secure because one does not know what "secure" means.

3. Which of the following demonstrate violations of the principle of least privilege? Please justify your answer.
   a. The UNIX *root* account?
   b. A user whose function is to maintain and install system software. This user has access to the source files and directories, access to only those programs needed to build and maintain software, and can copy executables into system directories for other users. This user has no other special privileges.

   Answer:
   a. As any process running as *root* can do anything, whether or not it be related to the particular task that the process is to perform, the *root* account demonstrates a violation of the privilege of least principle.
   b. In this case, the user can only perform system tasks related to the installation of software. Hence this does not demonstrate a violation of the principle of least privilege.

4. Consider the Bell-LaPadula multilevel security model. If a subject with security label (L, C) can read an object with security label (L', C'), then (L, C) is said to *dominate* (L', C'). Prove that this *dominates* relation is reflexive, antisymmetric, and transitive.

   Answer: A subject with security level (L, C) can clearly read an object at that same security level. Thus, (L, C) dominates (L, C), so dominates is reflexive. Suppose (L, C) dominates (L', C') and (L', C') dominates (L, C). Then a subject at (L, C) can read an object at (L', C') and a subject at (L', C') can read an object at (L, C). As reading up is disallowed, this means that the subject and object are at the same level, so $(L, C) = (L', C')$. Hence dominates is antisymmetric. Finally, suppose (L, C) dominates(L', C') and (L', C') dominates (*L*", *C*"). This means that a subject at level (L', C') can read an object at (L'', C''). So that subject can create an object at level (L', C') that contains the same information as the original object. But then a subject at (L, C) can read the new object, meaning that the subject can read the original object. Hence (L, C) dominates (L'', C''), proving transitivity.

5. Define each of the following terms in one short sentence:
   a. public key cryptosystem
   b. separation of duty
   c. ciphertext
   d. constrained data item
   e. principle of fail-safe defaults

   Answer:
   a. A public key cryptosystem is a cryptosystem with two keys, one of which is known to everyone (the *public key*) and the other of which is known only to one person (the *private key*).
   b. Separation of duty is the requirement that if two or more steps are required to perform a critical function, then at least two different people must perform them.
   c. Ciphertext is encrypted text.
   d. A constrained data item is an item to which the integrity constraints apply. It is used in the Clark-Wilson Integrity Model.
   e. The principle of fail-safe defaults states that access should be denied unless explicitly granted. It also requires that, should an operation be attempted and fail, the system's security state must be restored to the one before the operation was attempted.