# Answers to Sample Midterm

1. Represent a security compartment label using the notation

   <*security level*; *set of categories*>.

   Can a user cleared for secret; { dog, cat, pig } have read or write access (or both) to documents classified in each of the following ways under the military security model?

   a. <top secret; { dog }>

   b. <secret; { dog }>

   c. <secret; { dog, cow }>

   d. <secret; { moose }>

   e. <confidential; { dog, pig, cat }>

   *Answer*:

   a. No read (as top secret > secret); no write (as { dog, cat, pig } $\not\subseteq$ { dog })

   b. Read (as secret = secret and {dog} $\subseteq$ { dog, cat, pig }); no write (as { dog, cat, pig } $\not\subseteq$ { dog })

   c. No read (as { dog, cow } $\not\subseteq$ { dog, cat, pig }); no write (as { dog, cat, pig } $\not\subseteq$ { dog, cow })

   d. No read (as { moose } $\not\subseteq$ { dog, cat, pig }); no write{ dog, cat, pig } $\not\subseteq$ { dog, moose })

   e. Read (as confidential < secret and { dog, pig, cat } $\subseteq$ { dog, cat, pig }); no write (as confidential < secret)

2. 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
```

3. 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.

4. 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.

5. Consider a system that used the Bell-LaPadula model to enforce confidentiality and the Biba model to enforce integrity.
   a. If the security classes were the same as integrity classes, what objects could a given process (with some security class that also served as its integrity class) access?
   b. Why is this scheme not used in practice?

   *Answer*:

a. Assume the security classes were the same as the integrity classes. Let *A* and *B* be the labels of security compartments, where *A dom B*. Then under the Bell-LaPadula model, a subject with label *B* cannot read an entity with label *A*. Under Biba's model, a subject with label *A* cannot read an entity with label *B*. A similar set of conditions holds for writing. However, if $A = B$, then both models allow reads and writes. And, of course, if there is no dominance relation between any two labels, entities with those labels can neither read nor write one another. Thus, if the security classes are the same as the integrity class, a given process can only access objects in its own compartment (class).

b. This scheme is far too restrictive to be used in practice. The processes are completely confined to their compartments, and often processes need to be able to read data in compartments that their compartment dominates. This is not possible in this scheme.