Homework 4

Due: May 23, 2016 (Note this is a Monday) **Points:** 140

Questions

Remember to justify your answers.

1. (*20 points*) Given the security levels TOP SECRET, SECRET, CONFIDENTIAL, and UNCLASSIFIED (ordered from highest to lowest), and the categories A, B, and C, specify what type of access (read, write, both, or neither) is allowed in each of the following situations. Assume that discretionary access controls allow anyone access unless otherwise specified.

   (a) Paul, cleared for (TOP SECRET, {A, C}), wants to access a document classified (SECRET, {B, C}).
   (b) Anna, cleared for (CONFIDENTIAL, {C}), wants to access a document classified (CONFIDENTIAL, {B}).
   (c) Jesse, cleared for (SECRET, {C}), wants to access a document classified (CONFIDENTIAL, {C}).
   (d) Sammi, cleared for (TOP SECRET, {A, C}), wants to access a document classified (CONFIDENTIAL, {A}).
   (e) Robin, who has no clearances (and so works at the UNCLASSIFIED level), wants to access a document classified (CONFIDENTIAL, {B}).

2. (*20 points*) Alice and Bob are creating RSA public keys. They select different moduli n_{Alice} and n_{Bob}. Unknown to both, n_{Alice} and n_{Bob} have a common factor.

   (a) How could Eve determine that n_{Alice} and n_{Bob} have a common factor without factoring those moduli?
   (b) Having determined that factor, show how Eve can now obtain the private keys of both Alice and Bob.

3. (*100 points*) This problem asks you to write a program using C or C++ that will introduce you to come complexities of managing privileges, as well as obstacles you might encounter when doing computer security work.

   A computer science student is helping out on a project that involves monitoring a network. The student's job is to write a program that will pluck the URLs for all HTTP traffic from the network. This requires *root* privileges, but for policy reasons the professor who is running the project cannot give the student those privileges.

   The professor's solution is to create a small program, called *runpriv*, that will make a second program called *sniff*, that the student will write, setuid to *root*. This way, the student can write the program to get the URLs from the network, and execute it, without having *root* permissions and without asking a system administrator to make the program privileged at each iteration.

   The program *runpriv* works as follows:

   (a) Check that the student is running the program by comparing the real UID of the process with that of the student. (Assume you are the student for this testing.) If the test fails, print an error message and exit.
   (b) Prompt the user for his or her password, and validate it against the authentication credential in the UC Davis Central Authentication System (use the program *kinit*(1) for this). If the password entered is incorrect, print an error message and exit.
   (c) If the current working directory does not contain a file called *sniff*, print an error message and exit.
   (d) If *sniff* is not owned by the student, or is not executable by the owner of the file, or can be read, written, or executed by anyone else (except, of course, *root*), print an error message and exit. This step checks that the student owns the file; that the student can execute it; and that no-one else has any rights over it.
   (e) If *sniff* was created or modified over 1 minute ago, print an error message and exit.
   (f) Change the ownership of *sniff* to *root* (UID 0), its group to *proj* (GID 95), and its protection mode to 4550 (meaning setuid to owner, and only readable and executable by the owner and group members — when you call *chmod*(2), you *must* have the leading "0", or you will get strange results. For this exercise, you must use the *chown*(1) program to change the owner and group. This means you must execute that program from *runpriv*, giving the appropriate arguments. (In the CSIF, the command will fail because there is no group "proj" — just let the error message print, and continue.)

Your job is to write *runpriv*.

Please submit your program as described in the **All About Homework** handout. Don't forget to include a Makefile that will automatically compile your program. And as always, remember we will execute the program on the CSIF systems, so be sure it works there.

Your program must be robust! Out of the 100 points for this program, 50 will come from the robustness and security protections you add to it to keep it from being abused.

Extra Credit

4. (*20 points*) Prove that two users who perform a Diffie-Hellman key exchange will have the same shared key.