Homework 4

Due: May 24, 2013 at 11:55pm **Points:** 100

Questions

1. (*25 points*) Consider the RSA cipher with $p = 5$ and $q = 7$. Show that $d = e$ for all choices of public key d and private key e.

2. (*25 points*) Needham and Schroeder suggest the following variant of their protocol:

   1. Alice $\rightarrow$ Bob : Alice
   2. Bob $\rightarrow$ Alice : $\{$ Alice $||$ $rand_3$ $\}k_{\text{Bob}}$
   3. Alice $\rightarrow$ Cathy : $\{$ Alice $||$ Bob $||$ $rand_1$ $||$ $\{$ Alice $||$ $rand_3$ $\}k_{\text{Bob}}$ $\}$
   4. Cathy $\rightarrow$ Alice : $\{$ Alice $||$ Bob $||$ $rand_1$ $||$ $k_{session}$ $||$ $\{$ Alice $||$ $rand_3$ $||$ $k_{session}$ $\}k_{\text{Bob}}$ $\}k_{\text{Alice}}$
   5. Alice $\rightarrow$ Bob : $\{$ Alice $||$ $rand_3$ $||$ $k_{session}$ $\}k_{\text{Bob}}$
   6. Bob $\rightarrow$ Alice : $\{$ $rand_2$ $\}k_{session}$
   7. Alice $\rightarrow$ Bob : $\{$ $rand_2 - 1$ $\}k_{session}$

   Show that this protocol solves the problem of replay as a result of stolen session keys.

   Hint: Consider two cases: one in which the attacker does not send an initial message to Bob (that is, impersonate Alice in step 1), and the second where she does.
   (*text* §10.10, exercise 7).

3. (*50 points*) This problem requires you to do some source code analysis. To do this assignment, you need to download the file *lsu.tar* from the homework web site, copy it onto a system at the CSIF, and unpack it. Once on the CSIF, type

   ```
   tar xvf lsu.tar
   ```

   and you will find a directory named *lsu* containing the program *lsu*. It consists of several source files, a *Makefile*, and a large number of configuration files. The version you have downloaded is preconfigured to compile on the CSIF systems.

   This assignment consists of several steps. First, we'll run a source code analyzer over the program to find possible errors. Then, we will examine a couple of error reports.

   The program *sourceanalyzer* is a tool written by Fortify Software. It analyzes programs for possible vulnerabilities, and when it finds one, it describes it and gives a trace of the routines involved in an attack exploiting the vulnerability.

   First, run the program. See the document **The Fortify Source Code Analyzer** for details on how to do this. Remember, this program uses *make*, so you want to follow the directions in the section "More Complex Programs". The document also explains how to interpret the results.

   When you run the second command, be sure to save the output in a file; it's long, and you'll need to refer to it later.

   Now, look at the output. You will see many potential errors. We're going to focus on three.

   (a) First, search in the output for the string "BDA475DC463CF3193DA9048713AAD0C6"; you should see that the next line begins with "util.(249)". This indicates that there is an easily-exploitable ("high") vulnerability through which a user can supply any command ("Command Injection") to the *system*() library call at line 249 of the file *util.c*.

(b) Next, search in the output for the string "ADE01E7CEF8599C1929156D862BFD432"; the next line should begin with "syntax.c(61)". This indicates that there is a way to manipulate the path name of a file ("Path Manipulation") to have the *fopen*() function at line 61 of the file *syntax.c* open a file that you should not be able to, but that this requires some skill to do ("medium").

(c) Finally, search in the output for the string "12DDF45CCB1A271C8D91F4CDB8B33A13"; the next line should begin with "lsu.c(138)". This indicates that the system call *setuid*() at line 138 of *lsu.c*' makes a change in privileges ("Privilege Management") that is often misused. But this is hard to exploit ("low").

For the first two, describe in detail the flow of data that would allow an attacker to exploit the flaw. For example, for an integer overflow, you would say something like this:

1. Attacker passes $2^{31} - 1$ arguments to program (via main function in file main.c line 25)
2. Program appends 3 more arguments (see main.c line 36)
3. Counter for the loop in which the arguments are appended overflows (see main.c lines 34-38)

Of course, this is a made-up example, not one drawn from *lsu*. But it gives you the level of detail we want.

Please copy the messages from the run of *sourceanalyzer* describing the vulnerability and the trace of functions and files below it. Then write your description beneath.

For the third one, analyze the report. Is this an exploitable vulnerability? If so, explain how to do it; if not, please explain why it is not exploitable (or not a vulnerability).

Extra Credit

1. (*20 points*) Consider an RSA digital signature scheme. Alice tricks Bob into signing messages m_1 and m_2 such that $m = m_1 m_2 \bmod n_{Bob}$. Prove that Alice can forge Bob's signature on m.
(*text* §10.10, exercise 8).