# CS 161 Computer Security
# Spring 2010 Paxson/Wagner
# HW 1

## Due February 11, 11:59pm

**Instructions:** Submit your solution by Thursday, February 11, 11:59pm, in the drop box labelled CS161 in 283 Soda Hall. Print your name, your class account name (e.g., cs161-xy), your TA's name, the discussion section time where you want to pick up your graded homework, and "HW1" prominently on the first page. Staple all pages together. Your solutions must be legible and the solution to each problem must be labelled clearly. You must work on your own on this homework.

**1. (20 pts.) A simple web service**

You need to write a web service that accepts trouble reports via a web form and then forwards them to a system administrator. More specifically, the web service should take the text of a message written by the user explaining the problem and a name for the problem supplied by the user, and your program should email this information to `admin@mysite.com`. You do so by invoking the `mail` program and providing it the problem explanation on stdin, and the problem name in the email subject via a command-line argument.

Your find the following code on the web to do just this:

```
void send_mail(char *problem_report, char *problem_name)
{
  FILE *mail_stdin;
  char buf[512];
  sprintf(buf, "mail -s \"Problem: %s\" admin@mysite.com",
          problem_name);

  mail_stdin = popen(buf, "w");
  fprintf(mail_stdin, problem_report);
  pclose(mail_stdin);
}
```

Identify at least three security problems with this code. For each problem, provide or describe an example of an input that would demonstrate or illustrate the existence of the problem.

HINT: Familiarize yourself with the workings of `popen()` and `pclose()` if they are new to you. You can read the manual pages for `popen()` by typing `man popen` at a shell prompt on a Unix system.

**2. (20 pts.) Security principles**

Identify the security principle(s) relevant to each of the following scenarios, giving a one or two sentence explanation for each:

(a) At a closed event where the President will be speaking, there are security guards and metal detectors at the door, despite the presence of Secret Service agents throughout the hall.

(b) On trains, there is often a "dead man's switch" which must be pressed down at all times for the train to be in motion. If something were to happen to the driver, the switch would be released and the train's brakes would be applied.

(c) On overnight school trips, a teacher places masking tape on the outside of all the students' hotel room doors, connecting the door to the frame, so that in the morning the teachers will have a way to know if one of the students snuck out in the middle of the night without permission.

(d) San Francisco Muni buses require those who need to buy tickets to enter at the front. However, the rear doors are often left open for passengers to disembark, and people sometimes jump on at the back of the bus without paying the fare.

(e) Electronic music distributors sold their music with DRM (Digital Restriction/Rights Management) so that users were limited in how they could use the music. In response, some users either used software to break the DRM or turned to legally questionable sources for their music.

**3. (20 pts.) XSS**

Prof. Hinkley comes up with what he thinks is a great solution to the problem of cross-site scripting vulnerabilities. He suggests introducing a new HTML tag, `<NOJAVASCRIPT>`. In between `<NOJAVASCRIPT>` and `</NOJAVASCRIPT>`, JavaScript is disabled: browsers are supposed to ignore (and in particular, not execute) any JavaScript that may occur between these two tags. Prof. Hinkley suggests that web developers can use this to avoid cross-site scripting attacks: they should surround every place in their HTML page where they are including untrusted content with a `<NOJAVASCRIPT>` tag. For instance, consider the following vulnerable code:

```
w.write("Hello, " + name + "!  Welcome back.\n");
```

Because `name` comes from user input, the above code has a XSS vulnerability. Prof. Hinkley proposes that instead of writing code like the above, the web developer should use

```
w.write("Hello, <NOJAVASCRIPT>" + name
        + "</NOJAVASCRIPT>!  Welcome back.\n");
```

Similarly, instead of writing

```
w.write("Today's most popular link is: "
        + "<A HREF=\"" + url + "\">" + url + "</A>\n");
```

(which may be vulnerable, since `url` comes from user input), Prof. Hinkley proposes the web developer should write

```
w.write("Today's most popular link is: "
        + "<NOJAVASCRIPT><A HREF=\"" + url
        + "\">" + url + "</A></NOJAVASCRIPT>\n");
```

List at least two problems with Prof. Hinkley's proposal.

**4. (20 pts.) Spot the bug**

Here's a real security hole that occurred in `xterm`. At the time, the `xterm` application ran with permissions that let it read or write any file (for various reasons that aren't important here). `xterm` had a feature that allowed the user to enable logging, so they could log their terminal session to a file of their choice. Of

course, the user should only be allowed to enable logging to a file that the user has permission to write (for instance, we don't want to allow the user to overwrite the global password file, `/etc/passwd`). Therefore, `xterm` used the following code to ensure that the user has permission to write to the logfile, before writing to it:

```
if (access (logfile, W_OK) < 0)
    return ERROR;
fd = open (logfile, O_CREAT|O_WRONLY|O_TRUNC, 0666);
/* ... write to fd ... */
```

In this code, `access(logfile, W_OK)` checks file permissions on the specified file to see whether the user who launched `xterm` has permission to write to the specified file[1].

This code has a security vulnerability. What is it?

**5. (20 pts.) Cookies**

When a site sets a cookie on your browser, the cookie is typically associated with the domain of the server, or a set of domains. For instance, when I visit `http://www.paypal.com/`, Paypal sets a cookie on my browser that my browser will send back to the server any time I visit a page on `www.paypal.com`. However, the browser will not send this cookie to other third-party sites, like (say) `blogger.com` or `ha.ckers.org`, due to the *Same Origin Policy*. Similarly, pages from `www.paypal.com` can set this cookie to a different value at any time, but other third-party sites (like `blogger.com` or `ha.ckers.org`) cannot overwrite the value of this cookie.

(a) Why is it important that third-party sites must not be able to see the cookies set by Paypal? What could go wrong if they could?

(b) Why is it important that third-party sites must not be able to overwrite the cookies set by Paypal? What could go wrong if they could?

**6. (0 pts.) Optional: any feedback?**

Optionally, feel free to include feedback. What's the single thing we could to make the class better? Or, what did you find most difficult or confusing from lectures or the rest of class, and what would you like to see explained better?

Your answers will not affect your grade. Feel free to be frank: we appreciate all feedback, even (especially) critical feedback.

[1] If you're curious, `xterm` ran with its effective UID set to 0, i.e., root, and its real UID set to the userid of the user who launched `xterm`. The manual pages for `access()` and `open()` specify their behavior in a bit more detail. However you shouldn't need to know all this to answer the question