Homework 5: Exam Review

This homework is due **May 4, 2015** at **6 p.m.** and counts for 5% of your course grade. Late submissions will be penalized by 10% plus an additional 10% every 5 hours until received. Late work will not be accepted after 20.5 hours past the deadline. If you have a conflict due to travel, interviews, etc., please plan accordingly and turn in your homework early.

We encourage you to discuss the problems and your general approach with other students in the class. However, the answers you turn in must be your own original work, and you are bound by the Student Code. Solutions should be submitted electronically via SVN in plain text format by completing the template at the end of this document.

Concisely answer the following questions. Limit yourself to at most 80 words per subquestion.

1. **HTTPS.** A *self-signed certificate* makes the claim that a public key belongs to a particular server, without any trusted certificate authority (CA) to verify it. Browsers display a warning message when a site presents such a certificate, but users often override these warnings. Some websites use self-signed certs to avoid the trouble of obtaining a cert from a trusted CA.

   (a) Briefly explain how using HTTPS with a self-signed certificate provides protection against a passive eavesdropper.

   (b) How might a man-in-the-middle attacker compromise a site that uses a self-signed certificate, assuming that the client ignores browser certificate warnings?

   (c) Some sites use HTTPS with a certificate signed by a trusted CA for their login pages, then set a session cookie and use HTTP for the other pages on the site. Briefly compare the security of this design to the use, for all pages on the site, of:

      i. a self-signed certificate;

      ii. a certificate signed by a trusted CA.

2. **Web attacks.** Consider a fictitious social networking site called MyPlace. MyPlace has millions of users, not all of whom are particularly security-conscious. To protect them, all pages on the site use HTTPS.

   (a) MyPlace's homepage has a "Delete account" link which leads to the following page:

```
<p>Are you sure you want to delete your account?</p>
<form action="/deleteuser" method="post">
  <input type="hidden" name="user" value="{{username}}"></input>
  <input type="submit" value="Yes, please delete my account"></input>
</form>
```

   (The web server replaces {{username}} with the username of the logged-in user.)

   The implementation of /deleteuser is given by the following pseudocode:

```
if account_exists(request.parameters['user']):
    delete_account(request.parameters['user'])
    return '<p>Thanks for trying MyPlace!</p>'
else:
    return '<p>Sorry, ' + request.parameters['user'] + ', an error occurred.</p>'
```

   Assume that the attacker knows the username of an intended victim. What's a simple way that the attacker can exploit this design to delete the victim's account without any direct contact with the victim or the victim's browser?

   (b) Suppose that /deleteuser is modified as follows:

```
if validate_user_login_cookie(request.parameters['user'], request.cookies['login_cookie']):
    delete_account(request.parameters['user'])
    return '<p>Thanks for trying MyPlace!</p>'
else:
    return '<p>Sorry, ' + request.parameters['user'] + ', an error occurred.</p>'
```

   where validate_login_cookie() checks that the cookie sent by the browser is authentic and was issued to the specified username. Assume that login_cookie is tied to the user's account and difficult to guess.)

   Despite these changes, how can the attacker use CSRF to delete the victim's account?

   (c) Suppose that the HTML form in (a) is modified to include the current user's login_cookie as a hidden parameter, and /deleteuser is modified like this:

```
if request.parameters['login_cookie'] == request.cookies['login_cookie'] and
      validate_login_cookie(request.parameters['user'], request.cookies['login_cookie']):
    delete_account(request.parameters['user'])
    return '<p>Thanks for trying MyPlace!</p>'
else:
    return '<p>Sorry, ' + request.parameters['user'] + ', an error occurred.</p>'
```

   The attacker can still use XSS to delete the victim's account. Briefly explain how.

3. **Secure programming.** StackGuard is a mechanism for defending C programs against stack-based buffer overflows. It detects memory corruption using a *canary*, a known value stored in each function's stack frame immediately before the return address. Before a function returns, it verifies that its canary value hasn't changed; if it has, the program halts with an error.

   (a) In some implementations, the canary value is a 64-bit integer that is randomly generated each time the program runs. Explain why this prevents the basic form of stack-based buffer overflow attack discussed in lecture.

   (b) What is a security drawback to choosing the canary value at compile time instead of at run time? If the value must be fixed, why is 0 a particularly good choice?

   (c) No matter how the canary is chosen, StackGuard cannot protect against all buffer overflow vulnerabilities. List two kinds of bugs that can corrupt the stack and allow the adversary to take control, even with StackGuard in place.

4. **Ethics.** Consider the following scenario: A worm is infecting systems by exploiting a bug in a popular server program. It is spreading rapidly, and systems where it is deleted quickly become reinfected. A security researcher decides to launch a counterattack in the form of a defensive worm. Whenever a break-in attempt comes from a remote host, the defensive worm detects it, heads off the break-in, and exploits the same bug to spread to the attacking host. On that host, it deletes the original worm. It then waits until that system is attacked, and the cycle repeats.

   (a) Many people would claim that launching such a counterattack in this scenario is ethically unacceptable. Briefly argue in support of this view.

   (b) Are there circumstances or conditions under which an active security counterattack would be ethically justified? Briefly explain your reasoning.

Submission Template

You should create a directory in your SVN folder named "HW5". Within the "HW5" folder, you should place a file named "HW5.TXT". For example, I would create https://subversion.ews.illinois.edu/svn/sp15-cs461/mdbailey/HW5/HW5.TXT. Within "HW5.TXT" make sure each answer is formatted as a single line, that ALL the below formatting is exactly as shown below, and that the file you submit is in plain text format. Limit yourself to at most 80 words per subquestion.

```
# Problem 1

1a. [Answer ...]

1b. [Answer ...]

1c. [Answer ...]

# Problem 2

2a. [Answer ...]

2b. [Answer ...]

2c. [Answer ...]

# Problem 3

3a. [Answer ...]

3b. [Answer ...]

3c. [Answer ...]

# Problem 4

4a. [Answer ...]

4b. [Answer ...]
```