Homework 4: Web and Network Security

This homework is due **Friday, November 5** at **5 p.m.** and counts for 4% of your course grade. Late submissions will be penalized by 10% plus an additional 10% every 12 hours until received. (The professor may grant individual extensions, but only under truly extraordinary circumstances.)

You are free to discuss the problems with other members of the class, but the answers you turn in must be your own work. Email your submission (as text or as a PDF attachment) to `eecs398@umich.edu` with "`HW4 Submission`" as the subject. You should receive a confirmation email within 15 minutes.

---

1. **HTTPS.** A *self-signed certificate* is a 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 time and expensive 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 compromise a site that uses a self-signed certificate?

   (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. Compare the security of this design to the use, for all pages on the site, of (i) a self-signed certificate and (ii) a certificate signed by a trusted CA.

2. **Denial of service.** Denial-of-service (DoS) attacks attempt to overwhelm a server with a huge volume of requests. Researchers have proposed a defense against DoS attacks called *client puzzles*: The server provides a random challenge r and a difficulty parameter n, and the client has to produce a solution s such that SHA-256$(r||s)$ ends in n zero bits; clients must present a valid solution to receive service.

   (a) What is the expected number of hash computations for the client to compute the solution? How many hash computations does it take for the server to check the solution?

   (b) If an attacker enjoys an *amplification factor* of 64 (i.e., the attacker can cause the server to do 64 units of work by expending one unit of work), what should n be to negate this advantage?

   (c) Some denial-of-service attacks employ a large number of malicious clients to overwhelm the server. How can the system adjust the puzzles to ensure that legitimate clients receive service during such attacks without requiring them to do excessive work solving puzzles when the system is not under attack? Briefly relate your answer to supply and demand.

3. **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 the simplest way the attacker can exploit this design to delete the victim's account?

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

```
if user_is_logged_in(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>'
```

   (Assume 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
      user_is_logged_in(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. Explain how.

□