Computer Science 50   Week 8 Wednesday: October 27, 2010
Fall 2010   Andrew Sellergren
Scribe Notes

Contents

Computer Science 50   Week 8 Wednesday: October 27, 2010
Fall 2010   Andrew Sellergren
Scribe Notes

1   Announcements and Demos (0:00–11:00)

- This is CS50.

- 3 new handouts.

- Our apologies to you if you showed up for last night's office hours and found the waiting time extremely long. We actually overcompensated in moving our staff assignments to later in the week. We'll be available tonight and tomorrow night in large numbers and next week we'll adjust accordingly.

- If you've visited the course website recently, you may have noticed CS50 Chat making an appearance. It's our hope that this will be an asset to you as you watch lecture videos online so that you can communicate with other students doing the same thing. You might be happy to know that the fourth ever message posted to CS50 Chat was "WHERE IS MY SUPAH SUIT?!?"

- Apropos to this week's material is the release of Firesheep, a Firefox plugin that allows you to hijack sessions. Firesheep was developed by Eric Butler and Ian Gallagher to increase exposure of an inherent security flaw. Basically, if a website isn't using SSL to authenticate users, Firesheep allows you to impersonate others who login to that website on an unencrypted wireless connection. Check out codebutler.com for more details on how Firesheep works.

- Similar in spirit to Firesheep is Idiocy, a utility which sniffs for Twitter sessions and posts a tweet in their account letting them know that their account has been compromised.

- Another packet-sniffing utility is `tcpdump`, a command-line tool that allows you to listen promiscuously to all network traffic, meaning you'll be receiving packets that aren't just destined for your computer.

- Please know that just because it is possible to hijack someone's session doesn't mean you should! Even if there is nothing stopping you technologically from doing so on Harvard's campus, you still can face disciplinary consequences from the Ad Board!

2   Internet Security (11:00–34:00)

2.1   Wireless Encryption

- There are many different algorithms for wireless encryption, among them WEP, WPA, and WPA2. WEP is generally considered to be insecure since its secret key is only 5 characters long. After a few minutes of sniffing WEP-encrypted traffic, it is possible to brute-force crack it.

- So why doesn't Harvard implement one of these algorithms for encrypting its wireless networks? One barrier is the incompatibility of older laptops with newer encryption algorithms like WPA2.

- The inherent vulnerability with unencrypted wireless traffic derives from HTTP being a stateless protocol. What this means is that there is nothing built in to HTTP that will allow a web server to remember who you are. When you make a request to a web server and a connection is opened, you receive the data you requested and the connection is closed. When you access a different page on the same site, another connection must be opened. The web server doesn't inherently know that you're the same user who just recently opened another connection. Enter cookies. These are long alphanumeric strings that are assigned to users by a webserver after they login. Each time you access the website that you've logged into and received a cookie from, you send that cookie along with your HTTP request to identify you. Unfortunately, without SSL, this cookie is transmitted in the clear. Anyone on the same unencrypted wireless network can intercept that cookie and send it to the webserver in order to impersonate you.

2.2   SSL

- Of course, it's worth noting that even if you use an encrypted wireless network, your traffic is only encrypted between you and the first of some 30 routers between you and the web server.

- More secure, then, is SSL (secure socket later), which encrypts your web request for all the hops along the way to the web server and back. You can tell that SSL is being used if the URL begins with `https` instead of `http`. And, of course, the web server itself must support SSL. Facebook does, in fact, support SSL, but as soon as you login on the `https` version, it redirects you to the `http` version.

- There exists a Firefox plugin called Force-TLS which detects if you're requesting `http` on a site that supports `https`. It will then automatically rewrite the URL so that it begins with `https`.

- Unfortunately, even SSL is fallible. All it does is raise the bar slightly. If an adversary launches a man-in-the-middle attack, SSL is useless. That is, if an adversary positions himself between you and Facebook such that he intercepts your requests and sends them to Facebook on your behalf, he can present you with Facebook's login page and grab your login credentials when you submit them.

2.3   Virtual Private Networks

- What's interesting about many of these security issues is that they've existed since the internet was first created. Most of the solutions have, as

3

well, but they haven't been widely adopted.

- Virtual private networks (VPNs) are a more robust solution to man-in-the-middle attacks and session hijacking. With VPN, you establish a secure connection to Harvard's servers (or some other trusted servers) which then relay your requests to other web servers. In this way, all of your traffic is encrypted even if a third-party website doesn't support SSL.

2.4   Questions

- Question: does logging out help prevent session hijacking? Yes, in that it narrows the window during which an adversary could have stolen your cookie. However, if he already has stolen your cookie, logging out won't do much good, as he'll still be able to login as you if the cookie hasn't expired.

- Question: what is the point of Firesheep? There are always debates about whether security vulnerabilities such as this should be revealed widely to the public. Generally, one could argue it's a good thing for the information to be as widespread as possible given that the bad guys probably already know whereas most of the good guys, i.e. innocent users, aren't aware that they need to be protecting themselves. It will also theoretically force websites like Facebook to make SSL the default, as Gmail did not too long ago. The usual argument for not making SSL the default is cost: more CPU cycles are required to process SSL requests. However, Google was able to do it by increasing CPU cycles only by 2%. Hopefully, Facebook and others will soon follow suit.

- Question: will Firesheep work for everyone? Probably not, especially if your ethernet card doesn't support promiscuous mode, though most do.

2.5   Two-factor Authentication

- Two-factor authentication is another way of raising the security bar whereby a psuedorandom number generator, in the form of a physical device that fits on your keychain or via a text message to your cell phone, creates a several-digit passcode in sync with a server. Both the passcode and the regular password are then required to login. In this way, an adversary would need to hijack both your password and a physical device that you own in order to impersonate you.

3   More on Web Development (34:00–76:00)

- Take a look at the cheatsheets linked from the lectures page of the course website under Week 8. These will provide you with a quick albeit not exhaustive reference to PHP, MySQL, HTML, and CSS.

Computer Science 50   Week 8 Wednesday: October 27, 2010
Fall 2010   Andrew Sellergren
Scribe Notes

- Remember that all of the APIs which make Harvard data available are documented on the course wiki. Also documented are other Fun APIs such as TextMarks which allows you to receive user data via text messages as well as CDYNE and WebPurify which allow you to filter out profanity from your websites.

- In case you hear the terms in the coming weeks, know that Web 1.0 is a term for commerce-based websites like Amazon whereas Web 2.0 loosely refers to dynamic, user-driven websites like Facebook.

- So far, we've only examined how to create static websites using HTML. If we want to create dynamic websites whose content changes based on user input, we need to learn a programming language like PHP (or else Python, Ruby, Java, etc.).

3.1 Cookies

- If we open the Net tab in Firebug, we can watch as our web request is sent to Facebook when we login. It turns out that when we login to Facebook, we make more than one HTTP request. Separate requests are made for the various images, CSS, JavaScript, and HTML files that are combined to make a web page.

- If we click on the line that reads `POST login.php?login_a`, we see on the Post tab that our e-mail and password are transmitted in the clear. In the raw, the data we sent looks like the following:

  `&email=malan%40cs50.net&pass=123456`

  This is a URL-encoded string of parameters that we are passing via the POST method. Each key-value pair is separated by an ampersand and some of the characters, e.g. `%40`, are URL-encoded versions of special characters, e.g. `@`.

- HTML form data is neatly organized into key-value pairs which can then quite easily be turned into a hash table for you to use in your back-end language like PHP.

- When we've logged in successfully, Facebook will send back in the response headers a long alphanumeric string which we call a *cookie*. This cookie will be written to our hard drive and is only accessible by Facebook. When we access other pages on Facebook before this cookie's expiration date, we will send this cookie along with our web request so that Facebook knows who we are.

- Another response header of note is the `xs` parameter. This is the one that Firesheep grabs from unencrypted traffic in order to hijack sessions. Although it makes it much easier to do so, Firesheep does nothing more sophisticated than what we could do ourselves using a packet sniffer.

Computer Science 50   Week 8 Wednesday: October 27, 2010
Fall 2010   Andrew Sellergren
Scribe Notes

3.2   Intro to PHP

- To create our first PHP file, we can simply copy `hello.html` to `hello.php`. Incidentally, know that when `~username` appears in the URL, it points to your `public_html` directory, whereas when you're SSH'ed into the Cloud, `~username` refers to your home directory.

- PHP is an interpreted language, which means that it isn't translated into binary until the program is actually executed. This saves us a step during development, but adds a step during exection which means we suffer a hit to performance.

- When a `.php` file is requested from a web server, the web server first searches the file to see if there's any actual PHP code in it. If there is, it passes the code to the PHP interpreter, which will then execute statements that produce output for the browser. With our `hello.php`, we have no PHP source code, but the browser recognizes the HTML, so it displays it accordingly. This is one nice feature of PHP: it commingles seamlessly with HTML.

- Let's revert our source code and actually use some PHP to produce output:

```
<!DOCTYPE html>
<html>
    <head>
        <title>hello, world!</title>
    </head>
    <body>
        <?
            printf("hello, world!");
        ?>
    </body>
</html>
```

  The `<?` and `?>` are the short versions of the open and close PHP tags. You might see `<?php` written instead for the open tag, but we prefer the shorter, symmetric versions.

- Even this, however, isn't dynamic in its output. What if we print out the current time in seconds since 1970:

```
<?
    printf("hello, world!");
?>
```

Computer Science 50   Week 8 Wednesday: October 27, 2010
Fall 2010   Andrew Sellergren
Scribe Notes

3.3   Forms

3.3.1   `froshims1.php`

- `froshims1.php` mimics a site that David made years ago to help automate the process of registering for freshman intramural sports. There are form inputs for name, gender, captainship, and dorm. When we submit our choices, we can seem them as key-value pairs in the HTTP headers using Firebug, including "on" when the captainship checkbox is checked and "M" for gender when that radio button was selected.

- HTML forms can be submitted by one of two methods: GET and POST. Whereas GET transmits parameters via an addendum to the URL, POST transmits parameters via the HTTP headers. GET is slightly less insecure because it is more readibly visible, but it is also more convenient for creating persistent links for bookmarks. POST, of course, supports transmitting larger amounts of data (e.g. an image file) because URLs have a maximum length.

- In `froshims1.php`, we lay out the form using an HTML table:

```
<?
    /*************************************************************
     * froshims1.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.
     * Submits to register1.php.
     *************************************************************/
?>

<!DOCTYPE html>

<html>
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    <div style="text-align: center">
      <h1>Register for Frosh IMs</h1>
      <br><br>
      <form action="register1.php" method="post">
        <table style=
        "border: 0; margin-left: auto; margin-right: auto; text-align: left">
          <tr>
```

Computer Science 50   Week 8 Wednesday: October 27, 2010
Fall 2010   Andrew Sellergren
Scribe Notes

```
                    <td>Name:</td>
                    <td><input name="name" type="text"></td>
                  </tr>
                  <tr>
                    <td>Captain:</td>
                    <td><input name="captain" type="checkbox"></td>
                  </tr>
                  <tr>
                    <td>Gender:</td>
                    <td><input name="gender" type="radio" value="F"> F
                        <input name="gender" type="radio" value="M"> M
                    </td>
                  </tr>
                  <tr>
                    <td>Dorm:</td>
                    <td>
                      <select name="dorm" size="1">
                        <option value=""></option>
                        <option value="Apley Court">Apley Court</option>
                        <option value="Canaday">Canaday</option>
                        <option value="Grays">Grays</option>
                        <option value="Greenough">Greenough</option>
                        <option value="Hollis">Hollis</option>
                        <option value="Holworthy">Holworthy</option>
                        <option value="Hurlbut">Hurlbut</option>
                        <option value="Lionel">Lionel</option>
                        <option value="Matthews">Matthews</option>
                        <option value="Mower">Mower</option>
                        <option value="Pennypacker">Pennypacker</option>
                        <option value="Stoughton">Stoughton</option>
                        <option value="Straus">Straus</option>
                        <option value="Thayer">Thayer</option>
                        <option value="Weld">Weld</option>
                        <option value="Wigglesworth">Wigglesworth</option>
                      </select>
                    </td>
                  </tr>
                </table>
                <br><br>
                <input type="submit" value="Register!">
              </form>
            </div>
          </body>
        </html>
```

The `table` tag introduces a spreadsheet-type layout. If we wanted to

make this table actually visible, we can set its `border` CSS property to
a non-zero value. As it is, we're just using it to make sure things align
properly.

- We've implemented a radio button for the gender input by specifying
`type="radio"`. We can then have multiple options for this input and
they will be mutually exclusive (i.e. selecting one will deselect the others)
so long as all of the `input` tags have the same `name` attribute.

- The `select` tag implements a dropdown menu. We can specify how many
elements in the menu we want to show at a given time using the `size`
attribute. Each `option` tag that falls inside of the `select` tag represents
a choice in the dropdown menu. What is actually transmitted by the form
is stored in the `value` attribute for each of these `option` tags.

3.4 `register1.php`

- What happens to our form data once it's submitted? In the `action` at-
tribute of our `form` tag, we specified `register1.php`. Let's take a look at
its source code:

```
<?
    /***************************************************************
     * register1.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.  Redirects
     * user to froshims1.php upon error.
     ***************************************************************/

    // validate submission
    if ($_POST["name"] == "" || $_POST["gender"] == "" || $_POST["dorm"] == "")
    {
        header("Location: http://cloud.cs50.net/" .
               "~cs50/lectures/8/src/froshims/froshims1.php");
        exit;
    }

?>


<!DOCTYPE html>

<html>
  <head>
    <title>Frosh IMs</title>
```

```
      </head>
      <body>
        You are registered!  (Well, not really.)
      </body>
    </html>
```

If we try to submit our form data without filling in all the inputs, we
get bounced back to the first page. This is accomplished with the if
condition at the top of `register1.php`. Because we've submitted form
data using the POST method, all of that data is handed to us in a so-
called superglobal variable named `$_POST`. This variable is actually a hash
table or an associative array. In C, our arrays could only have numerical
indices, but in PHP, we have strings and many other types as indices
as well. In this if condition, we're checking if any of the indices in `$_POST`
have empty values associated with them. This would imply that the the
user failed to fill in one of the inputs. If that's true, then we redirect him
to the `froshims1.php` using the `header` function which spits out HTTP
headers.

- PHP supports multi-line comments enclosed by `/*` and `*/` or single-line
  comments beginning with `//`.

3.5 `register2.php`

- In `register2.php`, we provide the user with a more useful error message
  if he fails to provide us with some input:

```
<?
    /************************************************************************
     * register2.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.  Informs user of
     * any errors.
     ************************************************************************/
?>

<!DOCTYPE html>

<html>
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    <? if ($_POST["name"] == "" ||
```

10

```
                           $_POST["gender"] == "" ||  $_POST["dorm"] == ""): ?>
        You must provide your name, gender, and dorm!
        Go <a href="froshims2.php">back</a>.
      <? else: ?>
        You are registered!  (Well, not really.)
      <? endif ?>
    </body>
  </html>
```

The if condition is actually the same, but this time we provide an error
message and a link to go back.

3.6   register3.php

- In register3.php, we actually finally do something with the data if it's
  been correctly provided:

```
<?
    /**************************************************************************
     * register3.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.  Reports registration
     * via email.  Redirects user to froshims3.php upon error.
     **************************************************************************/

    // validate submission
    if ($_POST["name"] != "" && $_POST["gender"] != "" && $_POST["dorm"] != "")
    {

        $to = "malan@cs50.net";
        $subject = "Registration";
        $body = "This person just registered:\n\n" .
         $_POST["name"] . "\n" .
         $_POST["captain"] . "\n" .
         $_POST["gender"] . "\n" .
         $_POST["dorm"];
        $headers = "From: malan@cs50.net\r\n";
        mail($to, $subject, $body, $headers);
    }
    else
    {
        header("Location: http://cloud.cs50.net/" .
                "~cs50/lectures/8/src/froshims/froshims3.php");
```

```
        exit;
    }
?>

<!DOCTYPE html>

<html>
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    You are registered!  (Really.)
  </body>
</html>
```

After checking again if all the inputs have been provided, we proceed to
create an e-mail message, using the `.` as a concatenation operator, and
then sending the e-mail to `malan@cs50.net` using the `mail` function. Our
registration form finally works!