

Google™





Fun Hacks and Cool Javascript:

Techniques Behind the Google AJAX API Playground

Ben Lisbakken
5/27/09



Outline

- What is the AJAX API Playground?
- Javascript!
- A Few Playground Tricks
- XSRF Dangers
- Performance Optimization



AJAX API Playground



AJAX API Playground

- The Playground is an interactive Javascript code editor
- Its primary use is for showing AJAX API sample code
- All the code is open source
- It can be used to show any snippets of HTML/CSS/JS
- I made it because learning from documentation is tough and boring!
- <http://code.google.com/apis/ajax/playground/>

```

InteractiveSample.prototype.addShowHideClicks = function() {
    var i;
    for (i = 0; i < this.categories.length; i++) {
        var cat = this.categories[i];
        var catTitle = cat.childNodes[0];
        $(catTitle).bind('click', this.toggleShowHideSubCategories(cat));
    }

    for (i = 0; i < this.subCategories.length; i++) {
        var subCatTitle = this.subCategories[i].childNodes[0];
        $(subCatTitle).bind('click', this.toggleShowHideLIs(subCatTitle));
    };
};

```

```

InteractiveSample.prototype.createCategories = function() {
    // codeArray is from ajax_apis_samples.js
    this.selectCode = $('#selectCode').get(0);
    for (var i=0; i < codeArray.length; i++) {
        var category = codeArray[i].category;
        var container = null;
        var subCategory = null;
        var categoryDiv = null;
        var subCategoryDiv = null;
    }
};

```

Javascript!

```

        if (category.indexOf('-') != -1) {
            // that means that this category is a subcategorys
            var categorySplit = category.split('-');
            category = categorySplit[0];
            subCategory = categorySplit[1];
        }
    }
}

```

Why Talk Javascript

- Javascript is used in a huge majority of web apps
- Javascript is very easy and copy/pasteable but people generally don't know it very well
- There is a lot of coolness in Javascript
- I always wanted to attend a session that shows more advanced Javascript

WARNING

- If you are an expert in Javascript, you might get bored
- If you are a beginner in Javascript, you might get lost
- I'm showing techniques and tricks, I haven't made sure that the following code is perfect
- This is not copy/paste code, this is educational code.
- I just want to show some parts of Javascript that are magically delightful



Javascript Playground





Playground Tricks



Playground Tricks

- Executing arbitrary code safely
 - iFrames
- Adding Debug Bar and Firebug Lite in Output
 - Anonymous functions
 - `functionName.toString()`
- Breakpoints
 - Closure
 - Switching context trick for `console.log()`

Executing Arbitrary Code Safely

- Problem:
 - Executing arbitrary Javascript is really dangerous!
 - **Stay away from eval()**
- Solution:
 - Run arbitrary code in another domain that has no authentication and no access to secure resources

code.google.com

User writes code, clicks run

Code sent to other server



savedbythegoog.appspot.com

Saves code

Returns unique URL

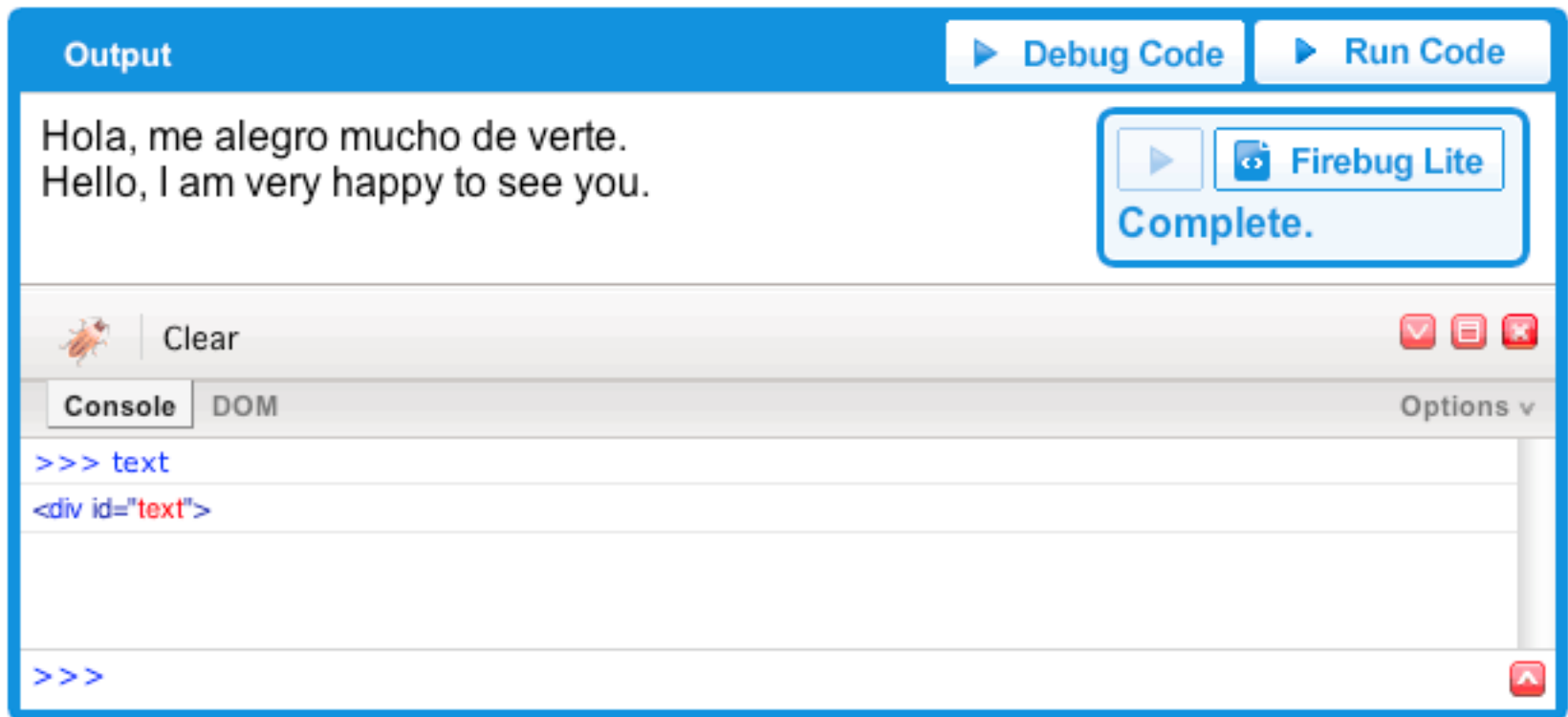


code.google.com

Creates iFrame to unique URL

Adding Debug Bar and Firebug Lite

- Need a way to dynamically insert a debug bar and Firebug Lite into the user's code if they click "Debug Code"



The screenshot displays a web application interface with a blue header bar. On the left, the word "Output" is written in white. On the right, there are two buttons: "Debug Code" and "Run Code", both with play icons. Below the header, the main content area shows two lines of text: "Hola, me alegro mucho de verte." and "Hello, I am very happy to see you." To the right of this text is a rounded rectangular box containing a play icon, the text "Firebug Lite", and "Complete." below it. Below the main content area is a console area with a grey background. It features a "Clear" button with a bug icon, a "Console" tab, and a "DOM" tab. The console shows the command ">>> text" and the output "<div id='text'>". There are also "Options" and "v" icons on the right side of the console. At the bottom of the console, there is a red arrow icon pointing up.

Adding Debug Bar and Firebug Lite

- Before sending user's code to the remote server, add Javascript in to initialize the Debug Bar and Firebug Lite
- Use anonymous function to keep it clean!
- Don't write the code as a string, it's too messy. Instead, use `.toString` on the function and have it autoexecute itself.

```
var anony = (function() {  
    function insertDebugBar() {  
        // insert debug bar!  
    }  
    function insertFirebugLite() {  
        // insert fblite!  
    }  
    insertDebugBar();  
    insertFirebugLite();  
});  
  
jsCode = "(" + anony.toString() + " ();" + jsCode;  
// send code to server to be saved, then iFramed
```



Adding Breakpoints

- No access to the Javascript engine, have to write in Javascript
- Need to preserve execution context
- Need to allow firebug to inspect local variables while in the breakpoint

```
10 content.innerHTML = '<div id="text">Hola, me alegro mucho de verte.</div><div id="translation"/>';  
11  
12  
13 // Grabbing the text to translate  
14 var text = document.getElementById("text").innerHTML;  
15  
16 // Translate from Spanish to English, and have the callback of the  
17 request  
18 // put the resulting translation in the "translation" div.
```

▶ Debug Code ▶ Run Code

▶ Firebug Lite
Paused (Line:14)

Adding Breakpoints

Before

```
7 function initialize() {  
8   // Grabbing the text to translate  
9   var text = document.getElementById("text").innerHTML;  
10  var a = text;  
11  alert(a);  
12 }
```

After

```
function initialize() {  
  // Grabbing the text to translate  
  var text = document.getElementById("text").innerHTML;  
  window.setContinue(false);  
  function breakpointAtLine9() {  
    if(!window.doContinue) {  
      if (window.logQueue) {  
        console.log(window.logQueue);  
      }  
      window.setTimeout(breakpointAtLine9, 100);  
    } else {  
      var a = text;  
      alert(a);  
    }  
  }  
  breakpointAtLine9();  
}
```




XSRF



XSRF

- Cross-site Request Forgery
- Common security hole
- When evil.com links to an action requiring authentication on good.com
- Without XSRF protection, if the user is still logged in to good.com, the action will be run

Example of XSRF

- You can save code in the Playground at <http://code.google.com/apis/ajax/playground/save>
- evil.com can create a POST form to <http://code.google.com/apis/ajax/playground/save>
- If a user is logged into the Playground and clicks this POST form, then evil.com will successfully save code for the user

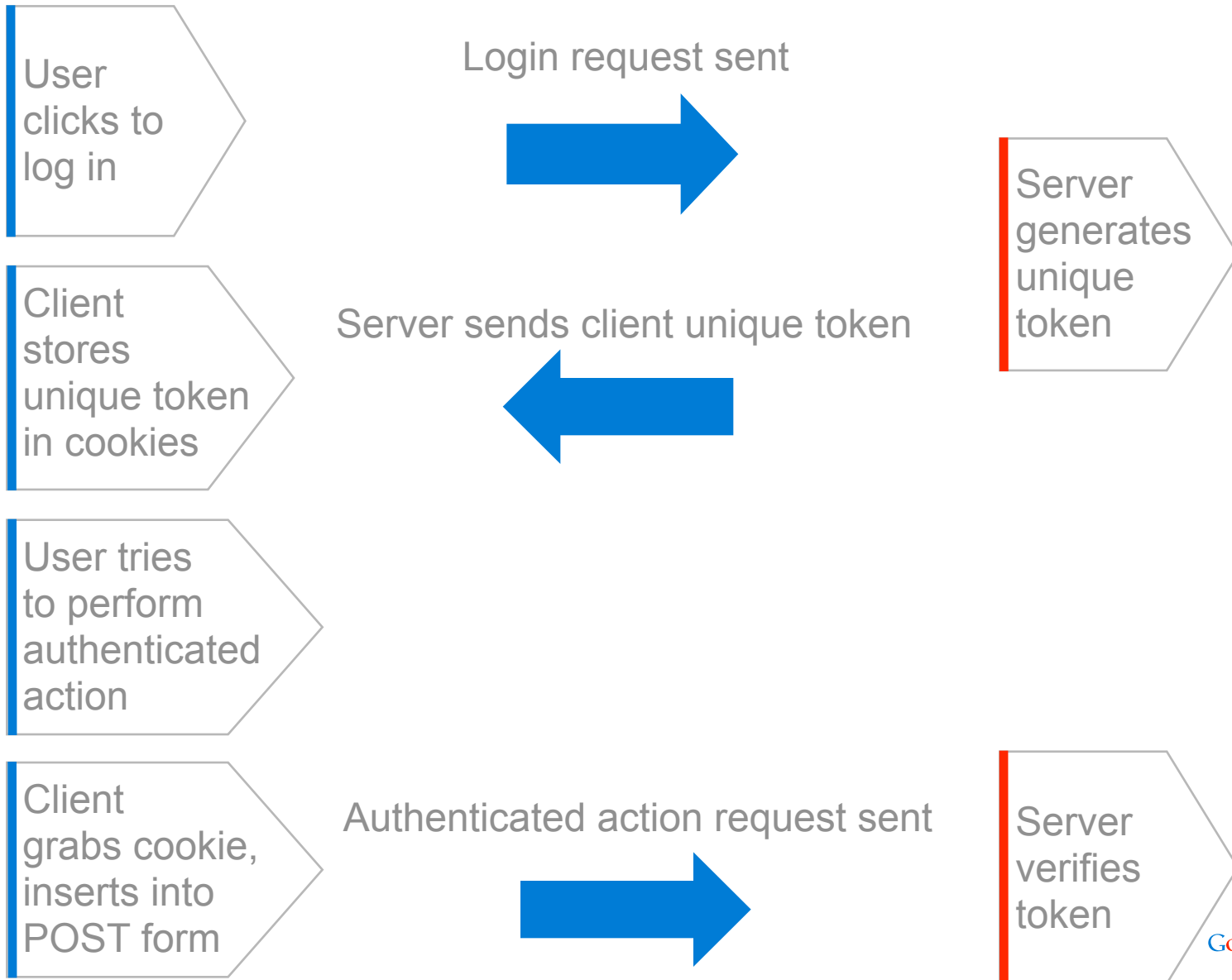
Why is XSRF Bad?

- Evil sites can make you access/modify private data on authenticated sites
- If bank.com had a form to let you reset your password to an e-mail address and the form was vulnerable to XSRF, then evil.com could force you to send them your password

(One way) XSRF Protection

- Only let known domains access/modify sensitive data
- Use a unique token to verify the origin of a request
- 1. When a user logs in, create a unique token on the server
- 2. Create a cookie on the client with this unique token
- 3. Whenever access a private resource, send this cookie from the client and check it on the server

XSRF Protection Diagram





Performance



Performance

- Performance is really important
- Google has done studies that indicate that you increase the performance of a site, you receive more traffic
 - **+500ms -> -20%** traffic
- Backend servers are fast
- If you want to make the user experience better, focus on the front-end because **80-90%** of end user response time is spent on the front end
- Go to [Steve Souders' blog!](#)

My Optimizations in the Playground

- By optimizing, I went from 400k -> 90k
- Page now loads in < 2 seconds
- I did several things:
 - gzipped responses!!
 - Minified Javascript
 - Correct script tag placement
 - Optimal cache headers
 - Minimized images

gzip

- The **biggest** space saver
- gzipping is compressing the content before sending it to the client (browser). The client unzips it on their side.
- Good because bandwidth is a bottleneck, but processing power to unzip isn't
- Current browsers accept gzip
- Commonly, you gzip JS/CSS/HTML/XML (text, not binary)
- You can get 70% space savings, or higher
- Easy to find instructions for setting gzip on common web servers
- App Engine has it on by default

Minify Javascript

- Reduce the number of unnecessary characters from Javascript (newlines, comments, tabs, spaces etc.)
- JSMIn, YUI Compressor, JS Packer, Dojo ShrinkSafe
- I use YUI Compressor, though all are good tools
- Good for production, bad for development
- Can save 20-30% of size

Script Tags

- Script tags can potentially be a giant bottleneck
- Avoid:
 - Lots of script tags
 - Script tags in the <HEAD>
- Do:
 - Put script tags at the bottom of <body>
 - Dynamically append script tags (if the scripts don't rely on each other)

Cache Headers

- Maximize browsers use of cache
- Set Expires header in the far future
- Be careful to version resources
- Set cache headers on the server
- Tools:
 - Fiddler, Wireshark, Firebug, Safari Network Timeline, YSlow, cURL

Minimize Images

- Images take up a lot of space
- Two ways to prevent this:
 - Compress images
 - Use Image Spriting



Questions?



Google™

