

# GREASEMONKEY HACKS™

*Tips & Tools for Remixing  
the Web with Firefox*



O'REILLY®

**Mark Pilgrim**  
*Foreword by Aaron Boodman,  
Creator and Lead Developer, Greasemonkey*

HACK  
#46

## Trace XMLHttpRequest Activity

Log XMLHttpRequest calls into JavaScript Console.

XMLHttpRequest is a JavaScript technique that enables a page to interact with the server without having to reload the entire page. This nonstandard API was first developed by Microsoft for Internet Explorer, but it was later picked up and implemented by most other browsers, including Firefox. Once used by only a few, it is now becoming more mainstream in the development of web applications.

The renewed interest for rich web applications such as Gmail, MSN Web-Messenger, and A9 Search, has crystallized a new nickname for the technique: *AJAX*.



Jesse James Garrett coined this term in early 2005 as a shorthand for “Asynchronous JavaScript And XML.”

A large number of frameworks now make use of the XMLHttpRequest object, trying to abstract the API and make it easier to use for a larger portion of hackers. However, as with most abstractions, there are still many times when you need to look under the hood.

Traditional debugging tools allow you to do that. You can install HTTP sniffers such as the LiveHTTPHeaders extension; you can test code interactively with the Venkman JavaScript debugger or the JavaScript shell; you can just litter your code with JavaScript `alert` statements. But these tools often offer too much or too little of what you actually need. This user script approaches debugging from a different angle, by focusing on the XMLHttpRequest interactions themselves and providing lightweight and instant tracing.

### The Code

A typical usage scenario of XMLHttpRequest starts with creating a new XMLHttpRequest instance, wiring some callbacks (such as `onreadystatechange`, `onLoad`, or `onerror`), and then calling `open` and `send`.

The basic approach of this script is to replace the `open` and `send` methods on any XMLHttpRequest instance that gets created. The replacement code mimics the behavior of the original methods, but it also traces the input parameters and adds some extra instrumentation on callback events.

Most common object-oriented languages differentiate the concept of *class* (the definition of an object) and *object* (an instance of a class).

Instead, JavaScript has classes only. When creating a new object, it uses another object as a template or prototype, rather than following an abstract blueprint (a class). It is called a *prototype-based language*.

This script takes advantage of this characteristic by modifying the prototype of the XMLHttpRequest constructor to replace the open and send methods on all XMLHttpRequest instances.

It overrides XMLHttpRequest.prototype.open and XMLHttpRequest.prototype.send with new implementations and keeps references to the original methods by backing them up into XMLHttpRequest.prototype.oldOpen and XMLHttpRequest.prototype.oldSend.



To keep the state of your UI, you often don't have to build your own structure in parallel with that of the document. The objects from the DOM can be extended with your own properties. In this case, the script uses the XMLHttpRequest object itself to store the unique ID for the object.

Because multiple calls to the server may occur simultaneously, using multiple XMLHttpRequest objects, it is useful to have an instance ID along with the traced information.

When first called on an XMLHttpRequest object, the uniqueID function will generate a random ID number and store it in the uniqueIDMemo property on the object. Subsequent calls will load that saved value and reuse it.



Greasemonkey lets you log events to JavaScript Console via the GM\_log method. That feature was added in Greasemonkey starting with Version 0.3, but didn't exist in earlier versions. In cases like that, you should test whether the feature is present and degrade gracefully if it is missing.

Save the following user script as *xmlhttprequest-tracing.user.js*:

```
// ==UserScript==
// @name           XMLHttpRequest Tracing
// @namespace      http://blog.monstuff.com/archives/cat_greasemonkey.html
// @description    Trace XMLHttpRequest calls into the Javascript Console
// @include        http://pick.some.domain
```

```
// ==/UserScript==

// based on code by Julien Couvreur
// and included here with his gracious permission

XMLHttpRequest.prototype.uniqueID = function() {
  if (!this.uniqueIDMemo) {
    this.uniqueIDMemo = Math.floor(Math.random() * 1000);
  }
  return this.uniqueIDMemo;
}

XMLHttpRequest.prototype.oldOpen = XMLHttpRequest.prototype.open;

var newOpen = function(method, url, async, user, password) {
  GM_log("[ " + this.uniqueID() + " ] intercepted open ( " +
    method + " , " +
    url + " , " +
    async + " , " +
    user + " , " +
    password + " )");
  this.oldOpen(method, url, async, user, password);
}

XMLHttpRequest.prototype.open = newOpen;

XMLHttpRequest.prototype.oldSend = XMLHttpRequest.prototype.send;

var newSend = function(a) {
  var xhr = this;
  GM_log("[ " + xhr.uniqueID() + " ] intercepted send ( " + a + " )");
  var onload = function() {
    GM_log("[ " + xhr.uniqueID() + " ] intercepted load: " +
      xhr.status +
      " " + xhr.responseText);
  };

  var onerror = function() {
    GM_log("[ " + xhr.uniqueID() + " ] intercepted error: " +
      xhr.status);
  };

  xhr.addEventListener("load", onload, false);
  xhr.addEventListener("error", onerror, false);

  xhr.oldSend(a);
}

XMLHttpRequest.prototype.send = newSend;
```



