



Your Short Cut to Knowledge

# Secure ASP.NET AJAX Development

Jason Schmitt

 Addison-Wesley  
Pearson Education

[www.awprofessional.com](http://www.awprofessional.com)

What This Short Cut Covers .....3

Section 1:  
AJAX, ASP.NET, and Atlas .....4


Section 2:  
AJAX Security Pitfalls.....19

Section 3:  
Securing ASP.NET AJAX.....44

Section 4:  
ASP.NET AJAX Security  
Testing .....81

About the Author .....92





Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this work, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this work, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Visit us on the Web: [www.awprofessional.com](http://www.awprofessional.com)

Copyright © 2007 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.  
Rights and Contracts Department  
75 Arlington St., Suite 300  
Boston, MA 02116  
Fax: (617) 848-7047

ISBN 0-321-49810-0

First release, November 2006

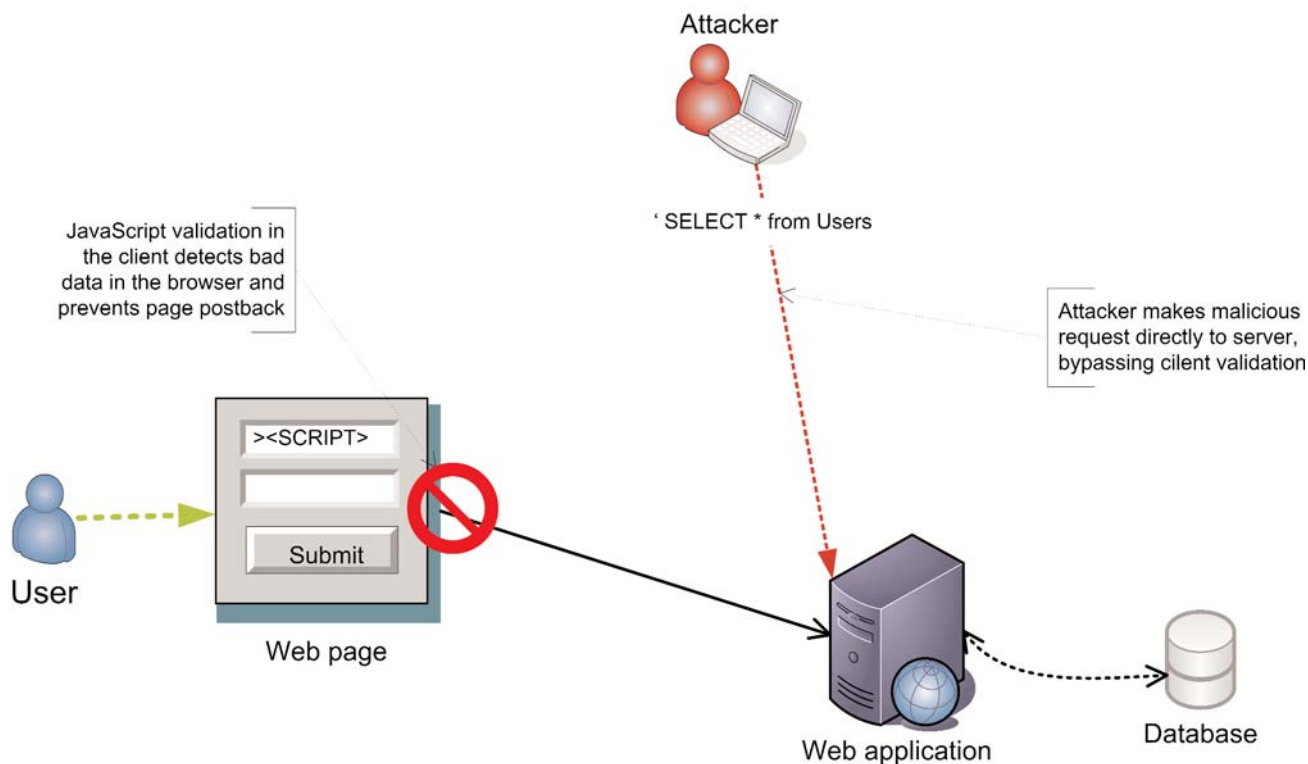
## SECTION 2

### AJAX Security Pitfalls

relying on client-side validation to protect you from malicious input. As pointed out in the last section, every bit of code that you send to the browser can be easily viewed and modified by an attacker. Not only can the attacker change what your client code is doing, but he also can skip it altogether and circumvent any client-side validation (see Figure 2-3).

**FIGURE 2-3**

Bypassing Client Validation



## SECTION 2

### AJAX Security Pitfalls

Here is how easy it is to bypass client validation. Download and install a client-side HTTP debugging proxy such as Fiddler.<sup>3</sup> Configure your browser to use this local proxy for Web requests, or in the case of Fiddler, just start up the tool and it auto-configures the proxy for you. Visit a Web page that accepts input from the user and submits it to the server. For an example, you can use the security testing FreeBank site available at <http://zero.webappsecurity.com>.<sup>4</sup> Enter valid information for all of the required fields on the page and submit the page. Now turn to Fiddler and view the request that you sent by selecting the login1.asp session in the View Session window. Switch to the Request Builder tab in Fiddler and drag your session from the View Session window into the Request Builder window. From this window, you can now modify anything you want in the request and send it to the server, bypassing any validation that existed in the client code. Figure 2-4 shows a request modified to include a SQL injection attack attempt. See the login parameter in the Request Body section for the malicious data going directly to the server.

From this example, you can see how easy it is to use proxy tools to bypass client validation, rendering it useless as a security countermeasure. With AJAX applications, you are increasing the number of inputs in your applications and you are using user-entered data to communicate directly with the server and Web services. Even if you are a diligent ASP.NET developer and are using validation server controls to validate all of your page inputs on the client and the server, you are still vulnerable with AJAX. If your application is taking data from a user and sending it directly to a Web service, exactly as we did in the stock quote ASP.NET AJAX example, you are opening up the possibility of completely bypassing validation if you are not validating every input into your Web service. ASP.NET validator server controls do not help you here. Validating the stock symbol in the client will save a trip to the server only if the data is invalid; it does nothing to protect the server from tainted data and potentially malicious commands.

<sup>3</sup> Fiddler:  
[www.fiddlertool.com](http://www.fiddlertool.com).

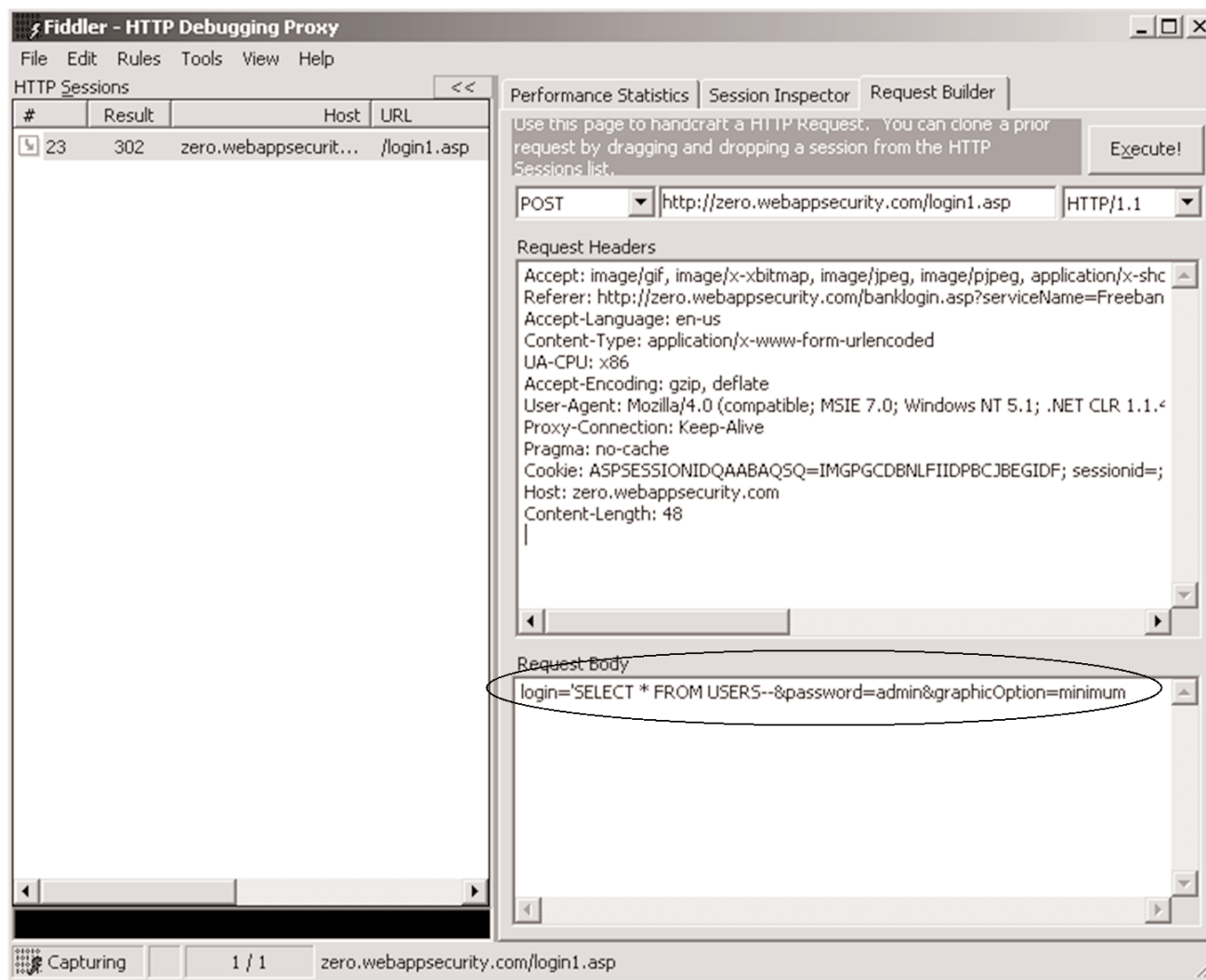
<sup>4</sup> Do not be fooled by the warning messages on the FreeBank site. The site is designed and maintained by a security vendor to appear like a real, hackable banking site for demo purposes.

## SECTION 2

### AJAX Security Pitfalls

**FIGURE 2-4**

Bypassing Client  
Validation in Fiddler  
to Attempt SQL  
Injection



## **Reliance on JavaScript**

One of the more difficult aspects of implementing AJAX applications is the tremendous amount of JavaScript programming that is usually required. If you refer back to the basic XMLHttpRequest example in Section 1, you see that AJAX in its most basic form is pure JavaScript programming. And even if the developer is pretty skilled with JavaScript programming, AJAX adds an additional layer of complexity by requiring you to have a fairly deep understanding of the DOM in the major browsers and the nuances of each browser's implementation of the XMLHttpRequest object.

The different AJAX client libraries, code generators, and frameworks that we outlined earlier reduce some of this burden of JavaScript complexity by abstracting a lot of the direct DOM and Web service manipulation and browser compatibility. But now you have to learn the JavaScript libraries provided to you by these development toolkits, so there is no getting around the fact that you cannot do AJAX without programming JavaScript. Even with ASP.NET AJAX, the most abstracted of any of these toolkits, you still have to write JavaScript. There is a *J* in AJAX for a reason, after all.

So where does the insecurity lie in JavaScript? Begin by reflecting on the fact that JavaScript was originally designed to do anything that a user can do, such as click on links or open and close windows. As an attacker, by design I can make a malicious script do everything a user can do, but in your browser. Like most other things we have talked about, JavaScript itself is not an insecure language. It is a very useful and powerful language, but it can do bad things if it is used incorrectly or manipulated by an attacker. Not only do you have to deepen your JavaScript skills to build secure AJAX applications, but also your users have to have JavaScript enabled in their browsers for your AJAX application to work. Some users just simply are not willing to do this, either through fear of script injection or because of corporate acceptable-use policies, but your AJAX application will not work otherwise.

So if JavaScript is not insecure, what is the problem? The security risk in JavaScript is twofold:

1. JavaScript has complete access to any information accessible in the browser, including HTML forms, page appearance, and content—everything in the DOM and the user's cookies.
2. Because JavaScript is downloaded and run in the client, it can be modified easily by an attacker.

Combine these risks with the fact that executable JavaScript code can be persisted in Web pages and run automatically in the background at predetermined intervals, all without the user knowing, and you can imagine all of the bad things that are possible with liberal use of JavaScript. Some Web services even send JavaScript functions back to the browser in clear text and the browser executes them without question. In fact, JavaScript on the client is integral to each of the remaining security pitfalls discussed in this section.

The only way to absolutely protect your application and your users from malicious JavaScript is to disable JavaScript in the browser, but this obviously is not practical in an AJAX application. What you need to do instead is to take precautions to ensure that you do not make sensitive resources available to client script and do not implement sensitive functions in vulnerable JavaScript code.

## **Cross-Site Scripting**

Cross-site scripting (see Figure 2-5) is one of the more dangerous security vulnerabilities in Web applications and has become the avenue of choice for attackers in recent years. Cross-site scripting describes a vulnerability whereby an attacker is able to execute malicious script code in a user's browser to gather data from the user. Through this ability to run script on someone else's client browser, the attacker can do a wide variety of bad things, including stealing cookies and hijacking

## SECTION 2

### AJAX Security Pitfalls

**FIGURE 2-5**  
Cross-Site Scripting  
with AJAX

