

Ajax security dangers

White paper



Table of contents

Introduction	3
Problems with traditional web applications	3
Ajax web applications	3
Security issues for Ajax web applications	4
Increased attack surface	4
Information leakage	4
Repudiation of requests and cross-site scripting	5
Ajax bridging	6
Hype about Ajax	7
Recommendations	7

Introduction

With the growth of Web 2.0 and the movement towards a more interactive and responsive web, many applications now employ Ajax, a collection of technologies that has received a large amount of publicity and is being quickly adopted by developers and corporations. While Ajax can greatly improve the usability of a web application by improving its responsiveness and flexibility, it can also create several attack opportunities if the application has not been designed with security in mind. This white paper describes the difference between traditional and Ajax web applications, the benefits of the Ajax methodology, its security concerns and recommendations for securing web application development.

Problems with traditional web applications

In traditional web applications, the browser typically refreshes the entire web page when it receives a response from the server. This design often results in long pauses from when the client sends a request to the Internet to when the server returns a response, negatively affecting performance.

Ajax web applications

Ajax is an acronym for Asynchronous JavaScript and extensible markup language (XML). It is not a programming language or technology but a grouping of other technologies that improves web application performance.

In Ajax web applications, the response time between the client request and the server response is reduced.¹ This reduction is accomplished by exchanging small amounts of data between the user's browser and the server without refreshing the web page with each response. This design can drastically improve response time.

In Ajax applications, JavaScript plays a larger role than in traditional web applications. At the beginning of a session, the browser loads an Ajax engine, usually written in JavaScript, along with a web page. The engine then displays the web page that the user sees and communicates the user's requests back to the server. The presence of the Ajax engine lets the user interact with the application without constantly refreshing the page.²

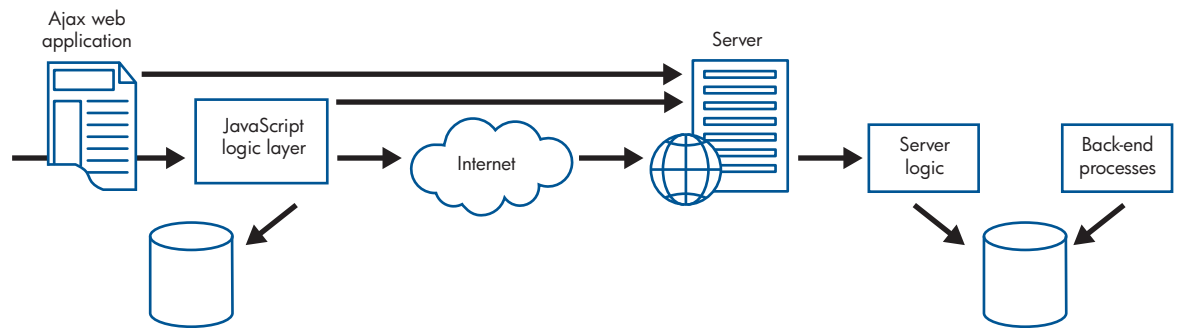
The Ajax engine sends Hypertext Transfer Protocol (HTTP) requests to the server asynchronously. Therefore, the application continues to respond to user events and interaction while the server processes the request. When the client receives the response, the engine manipulates the document object model (DOM) to present the results to the user.

Figure 1 shows the Ajax web application model.

¹ The actual time between the client request and the server response only appears to be reduced. Ajax does not affect the server-side processing time. Its benefit is that Ajax provides the appearance of a more responsive website.

² For more information about Ajax, see the essay, *Ajax: A New Approach to Web Applications*, from adaptive path.

Figure 1: Ajax web application model



Security issues for Ajax web applications

While Ajax can greatly improve the usability of a web application, it can also create several opportunities for possible attacks if the application is not designed with security in mind. As Ajax web applications exist on both clients and servers, they include the following security issues:

- Creating a larger attack surface with more inputs to secure
- Exposing internal functions of the web application server
- Allowing a client-side script to access third-party resources with no built-in security mechanisms

This section describes these security concerns in more detail.

Increased attack surface

Unlike traditional web applications, which exist almost entirely on the server, Ajax applications extend across the client and server. Such implementations require a trust relationship between the client and server—a relationship that attackers can exploit.

You can compare a traditional web application to a two-story house with no windows and only two ways to get in: a front door and a back door. Attackers have only two ways to break into the house.

An Ajax web application, however, sends many small requests, which create more inputs into the application. These inputs, sometimes called Ajax endpoints, provide more ways for attackers to get into the application. Using the house analogy, the two stories now have windows in addition to the front and back doors. The doors can be locked, but attackers now have many windows that they can break into and enter. The doors are no longer of consequence.

Information leakage

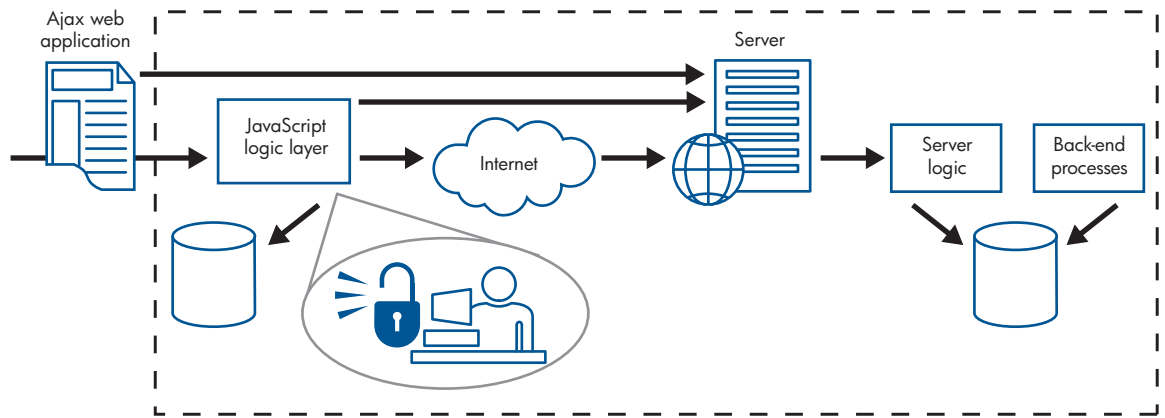
The JavaScript in the Ajax engine traps the user commands and sends function calls to the server. The following are examples of user commands:

- Return the price for product ID 24.
- Return valid cities for a given state.
- Return the last valid address for user ID 78.
- Update a user's age in a database.

Function calls provide “how to” information for each user command. This information places an attacker inside the application. From this vantage point, the attacker possesses function names, variable names, function parameters, return types, data types and valid data ranges.

Figure 2 shows an attacker virtually inside an Ajax web application.

Figure 2: An attacker is virtually in an Ajax application.



Repudiation of requests and cross-site scripting (XSS)

Browser requests and Ajax engine requests look identical. The server cannot discern a request made by JavaScript and a request made in response to a user action. Therefore, it is difficult for users to prove that they did not perform certain actions.

Using Ajax, JavaScript can also send a request for resources in the background without users knowing. The browser automatically adds the necessary authentication or state-keeping information, such as cookies, to the request. JavaScript code can access the response to the hidden request and send additional requests. This expansion of JavaScript functionality increases the possible damage of a cross-site scripting (XSS) attack.

XSS is the injection of script, such as JavaScript or VBScript, into the page that a user's browser receives. The script is then executed from the browser, exposing the user to a variety of threats, such as cookie theft (session hijacking and information leakage), keystroke logging, screen scraping and denial of service attacks.

Ajax amplifies XSS

With Ajax, XSS can make malicious requests with a user's credentials without refreshing the web page. Such activity can increase the potential for damage and theft of information. With traditional web applications, most information thefts occur with passive screen scraping. The malicious script can secretly read the contents of the HTML only from the page that the user is currently viewing. This content can then be sent to the attacker. Using Ajax, an XSS attack can send requests for pages other than the page that the user is currently viewing. This lets the attacker actively "hunt" for certain content and potentially access resources and data not available through passive scraping.

Injecting and propagating XSS

With traditional web applications, attackers manually inject and propagate XSS. Attackers can typically inject only one part of a website with reflection-based XSS, such as including scripts in an e-mail link, or stored XSS, such as saving the script as an entry in a back-end database.

With Ajax applications, XSS can propagate like a virus. The XSS payload can use Ajax requests to automatically inject itself into pages and reinject the same host with more XSS. This can occur with no hard refresh. Thus, XSS can send multiple requests using complex HTTP methods to propagate itself invisibly to the user. Several examples have already surfaced.

In October 2005, a member of MySpace.com created a profile that included a self-propagating worm, known as the Samy worm, making it the first public use of XSS and Ajax. When a user viewed the profile, the worm code automatically added the viewer to Samy's "friends" list. The worm code was also copied into the victim's profile, so that when another user viewed the victim's profile, the infection spread.

In June 2006, the Yamanner worm infected Yahoo's popular web mail service. The worm, using XSS and Ajax, took advantage of a vulnerability in Yahoo mail's onload event handling. When a user opened an infected e-mail, the worm code executed its JavaScript and sent a copy of itself to the infected user's Yahoo contacts. The infected e-mail carried a spoofed 'From' address that was randomly added from the infected system, making the malicious e-mail appear that it came from a known source.

It took only eight months for XSS and Ajax worms to develop from proofs of concept to full-fledged, malicious attacks. Attackers are aware of this vulnerability.

Ajax bridging

For security purposes, Ajax applications can only connect to the websites from which they come. For example, JavaScript with Ajax downloaded from Yahoo cannot connect to Google. To contact third-party sites, the Ajax service bridge was created. In a bridge, a host provides a web service that acts as a proxy to forward traffic between the JavaScript running on the client and the third-party site. A bridge provides a “web service to web service” connection. Many web frameworks, such as Microsoft® ASP.NET AJAX, support Ajax bridging. Custom solutions using PHP or common gateway interface (CGI) programs can also provide bridging.

An Ajax bridge can connect to any web service on any host, using protocols such as Simple Object Access Protocol (SOAP) and representational state transfer (REST). Bridges can also connect to a custom web service and arbitrary web resources, such as rich site summary (RSS) feeds, HTML, Flash and binary content.

Security issues with bridges

Ajax bridges introduce several major security issues. For example, an Ajax-enabled, online book store called securitybooks.com wants to access some of the web services that majorbookstore.com provides, such as an author search or genre recommendation service. To do this, securitybooks.com sends JavaScript code over an Ajax bridge to majorbookstore.com. Although anyone can get a free account to access majorbookstore.com’s web services, the free accounts have limited privileges: The number of unique queries, simultaneous queries and hits per second is set low. A formal partnership agreement between the two companies allows securitybooks.com to access majorbookstore.com with fewer restrictions.

In this example, an attacker can do several things. To copy the entire author database from majorbookstore.com, the attacker can issue thousands of queries to the Ajax bridge running on securitybooks.com. The relationship between the two websites allows the attacker to extract more data by going through securitybooks.com than by using a free account from majorbookstore.com. securitybooks.com can cache the results it receives from majorbookstore.com to limit the number of queries it has to make, reduce bandwidth and improve performance. As the attacker’s query may be in the cache already, the attacker may be able to extract data faster by using securitybooks.com.

An attacker can also send malicious requests through the Ajax bridge from securitybooks.com to majorbookstore.com. The bridge is another layer for the attacker to hide behind. Normal business transactions can also be hampered if an intrusion prevention system (IPS) at majorbookstore.com detects the malicious requests coming from the securitybooks.com IP address and then automatically blocks all requests from securitybooks.com. An attacker may try to launch a structured query language (SQL)SQL injection or an XSS attack against majorbookstore.com but instead cause a denial of service attack against all securitybooks.com users.

The majorbookstore.com may not detect the attack being relayed through the Ajax bridge. This may happen if majorbookstore.com does not review the requests it receives from securitybooks.com for malicious content as closely as the requests from other sites. The two parties have a partnership, and significant traffic comes to marjorbooks.com from securitybooks.com. In this case, majorbooks.com web services are vulnerable only if the attacks use securitybooks.com as an intermediary.

Hype about Ajax

Ajax and its effects on the Internet have received much publicity recently. Companies with complex web applications, such as MySpace.com, del.icio.us, and Writely.com, have been purchased for millions of dollars. Many conferences about Ajax and Web 2.0 applications have occurred all over the world. The hype about Ajax, however, has its own security ramifications.

First, the hype is driving programmers into the web application field. These novices often use “cut-and-paste” application development to add functionality to their projects without fully understanding the potential consequences. Most of these programmers are unfamiliar with the security nuances of developing web applications. This influx of programmers decreases the knowledge of those in the field. In turn, more technical articles, computer books and online forums are targeted at novices. Ultimately, a small number of experts can provide authoritative answers to an expanding audience.

Second, the demand for these sophisticated web applications is causing some companies to embrace client-side web applications. Often companies want to appear state of the art, fend off start-up competitors or provide new front-end capabilities that their users will enjoy. Many of these companies do not consider whether their applications should be web applications, let alone Ajax web applications.

Recommendations

If you are considering converting a business application into a web application, ask the following questions:

- Should the application be a web application?
- What do you gain by the conversion?
- Should the web application be an Ajax web application?

If you decide to develop a web application, whether traditional or Ajax, follow these recommendations:

- Document the inputs that are allowed in your application.
- Validate all input before processing it.
- Use white listing rather than black listing for validation. White listing means you accept what you know is good data, while black listing uses a list of data that is not allowed. For example, you know that a ZIP code should always have five numbers. White listing the ZIP code input means only accepting five numbers, nothing else.
- Always use the Secure Socket Layer (SSL) protocol to transmit sensitive data, such as log-in credentials or bank account numbers.

If you decide to retrofit your current web application or design a new one with Ajax, you should reduce the exposed program logic.

To learn more, visit www.hp.com/go/software

© Copyright 2007 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Microsoft and Windows are U.S. registered trademarks of Microsoft Corporation.

4AA1-5388ENW, October 2007

