

Ajax Security Dangers

By Billy Hoffman, SPI Labs

Ajax Security Dangers

Table of Contents

<i>Introduction</i>	3
<i>Problems with Traditional Web Applications</i>	3
<i>Ajax Web Applications</i>	3
<i>How Ajax Works</i>	4
<i>Security Issues for Ajax Web Applications</i>	5
<i>Increased Attack Surface</i>	5
<i>Information Leakage</i>	6
<i>Repudiation of Requests and Cross-Site Scripting</i>	7
<i>Ajax Bridging</i>	10
<i>Hype Surrounding Ajax</i>	12
<i>Recommendations</i>	14
<i>Footnotes and References</i>	15
<i>About SPI Labs</i>	16
<i>About S.P.I. Dynamics Incorporated</i>	17
<i>Contact Information</i>	18

Ajax Security Dangers

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 enthusiastic publicity from the media and quick adoption by developers and corporations alike. 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 whitepaper 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 Web page is typically refreshed in the browser only when a response is received from the server. This design often results in long pauses between when the request is sent over the Internet by the client and the response is subsequently returned by the server, 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 together improve Web application performance.

Ajax Security Dangers

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 entire Web page with each response. This design can drastically improve response time.

How Ajax Works

In Ajax applications, JavaScript takes 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. This engine then displays the Web page that the user sees in the browser, and communicates the user's requests back to the server. The presence of the Ajax engine allows the user to interact with the application without constant interaction with the server. [2]

The Ajax engine sends HTTP requests and provides immediate feedback to the user. The application, therefore, continues to respond to user events and interaction. When the response is received from the server, the engine manipulates the Document Object Model (DOM) to present the results to the user.

The following graphic shows the Ajax Web application model.

Ajax Security Dangers

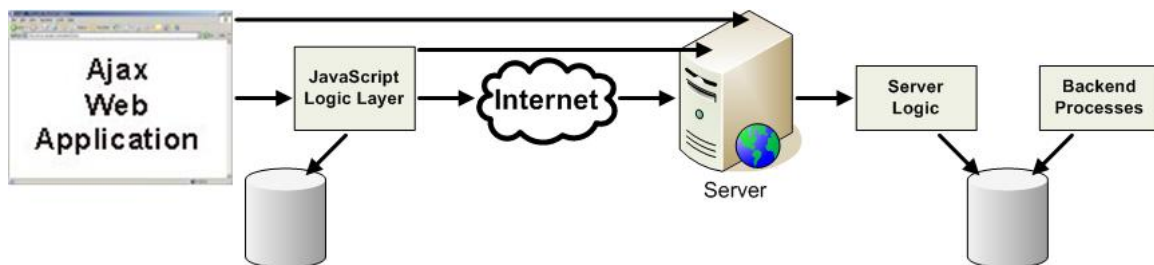


Figure 1: Ajax Web Application Model

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 attack if the application is not designed with security in mind. Since Ajax Web applications exist on both the client and the server, they include the following security issues:

- Create a larger attack surface with many more inputs to secure
- Expose internal functions of the Web application server
- Allow a client-side script to access third-party resources with no built-in security mechanisms

The following paragraphs describe these security concerns in more detail.

Increased Attack Surface

Unlike traditional Web applications that 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 can be exploited by an attacker.

Ajax Security Dangers

Think of a traditional Web application as a two-story house with no windows and only two ways to get in: a front door and a back door. There are only two ways for an attacker to break into such a 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 an attacker to get into the application. Returning to the house analogy, in addition to the front and back doors, there are now two stories filled with windows. The doors may be locked tight, but an attacker now has a multitude of windows that can be broken and entered at will. The doors are no longer of consequence.

Information Leakage

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

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

Function calls provide "how to" information for each user command that is sent. This information, being sent in clear text, in essence places the attacker

Ajax Security Dangers

inside the application. From this vantage point, the attacker possesses function names, variable names, function parameters, return types, data types, and valid data ranges.

The following graphic shows an attacker virtually inside the Ajax Web application.

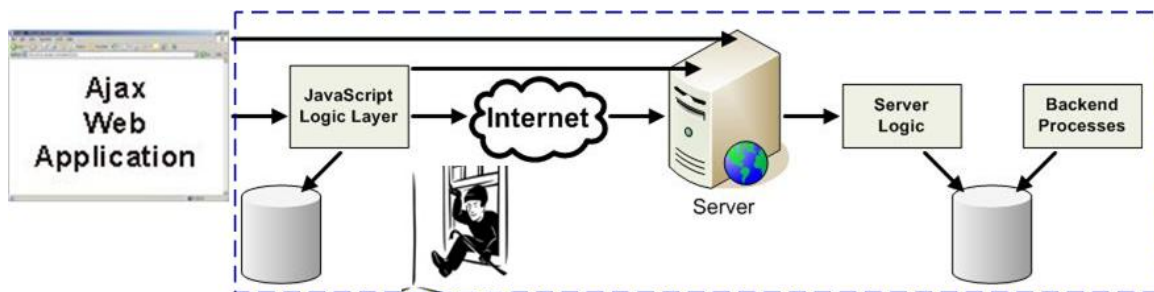


Figure 2: Attacker Virtually In Application

Repudiation of Requests and Cross-Site Scripting

Browser requests and Ajax engine requests look identical. The server is incapable of discerning a request made by JavaScript and a request made in response to a user action. This fact means it is very difficult for an individual to prove that they did not do a certain action.

It also means that JavaScript can make a request for a resource using Ajax that occurs in the background without the user's knowledge. The browser will automatically add the necessary authentication or state-keeping information such as cookies to the request. JavaScript code can then access the response

Ajax Security Dangers

to this hidden request and then send more 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 is returned to the user's browser. The script is then executed from the user's 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 drastically increases the damage and theft of information capable through XSS. With traditional Web applications most information theft occurred with passive screen scraping. The malicious script can secretly read the contents of the HTML only from the page the user is currently looking at. This content could then be sent back to the attacker. Using Ajax, a XSS attack could send requests for specific pages beside the page the user is currently looking at. This allows the attacker to actively "hunt" for certain content, potentially accessing resources and data not available to passive scraping.

Injecting and Propagating XSS

With traditional Web applications, injecting and propagating XSS was done manually by an attacker. Attackers could typically inject only one part of a

Ajax Security Dangers

Web site with reflection-based XSS, such as when scripts are included in a link in an e-mail, or stored XSS, such as when the script is saved as an entry in a backend database.

With Ajax applications, XSS can propagate like a virus. The XSS payload can use Ajax requests to autonomously inject itself into pages, and easily re-inject the same host with more XSS, all of which can be done with no hard refresh. Thus, XSS can send multiple requests using complex HTTP methods to propagate itself invisibly to the user. Several examples of this 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 the profile was viewed, 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 the victim's profile was viewed, 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 an infected email was opened, the worm code executed its JavaScript, sending a copy of itself to all the yahoo contacts of the infected user. The infected email carried a spoofed 'From'

Ajax Security Dangers

address picked randomly from the infected system, making the malicious email appear to come from a known source.

Note that it took only eight months for XSS and Ajax worms to develop from a proof-of-concept to a full-fledged malicious attack. Attackers are fully aware of this vulnerability.

Ajax Bridging

For security purposes, Ajax applications can only connect back to the Web site from which they come. For example, JavaScript with Ajax downloaded from yahoo.com cannot make connections to google.com. To allow Ajax to contact third-party sites in this manner, 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 could be considered a "Web service to Web service" connection. Many Web frameworks, such as Microsoft's "Atlas," provide support for Ajax bridging. Custom solutions using PHP or Common Gateway Interfaces (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) or Representational State Transfer (REST). Bridges can also connect to a custom Web service or even arbitrary Web resources such as Rich Site Summary (RSS) feeds, HTML, Flash, or even binary content.

Ajax Security Dangers

Security Issues with Bridges

Ajax bridges introduce several major security issues. Assume, for instance, that an Ajax-enabled online book store called spibooks.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, spibooks.com sends JavaScript code over an Ajax bridge to majorbookstore.com. While anyone can sign up for a free account to access majorbookstore.com's Web services, these free accounts have very limited privileges: the number of unique queries, number of simultaneous queries, and number of hits per second will be set very low. A formal partner agreement between the two companies allows spibooks.com to access majorbookstore.com with fewer restrictions.

In this scenario, an attacker can do several things. If the attacker wants to copy the entire author database from majorbookstore.com, he or she can simply issue thousands of queries to the Ajax bridge running on spibooks.com. The relationship between the two Web sites allows the attacker to extract more data by going through spibooks.com than if he or she had used a free account directly from majorbookstore.com. It is common in these situations for spibooks.com to limit the number of queries it has to make, reduce bandwidth, and improve performance for its users by caching the results it receives from majorbookstore.com. Since the attacker's query may already be in the cache, the attacker may be able to extract data faster by using spibooks.com.

Ajax Security Dangers

An attacker can also send malicious requests through the Ajax bridge from spibooks.com to majorbookstore.com. At the very least the bridge is another layer for the attacker to hide behind. Moreover, normal business transactions may be hampered if an Intrusion Prevention System (IPS) at majorbookstore.com detects the malicious requests coming from spibooks.com's IP address, and then automatically blocks all requests from spibooks.com. An attacker, for example, may attempt to launch a SQL Injection or an XSS attack against majorbookstore.com, but instead may cause a Denial of Service attack against all spibooks.com users.

It is also possible that majorbookstore.com will not detect the attack being relayed through the Ajax bridge. This may happen if majorbookstore.com does not scrutinize the requests it receives from spibooks.com for malicious content as closely as the requests it receives from others. This is common practice, since the two parties have an agreement to help each other and there is an immense amount of traffic coming in from spibooks.com. In this situation, majorbooks.com's Web services are vulnerable only if the attacks are sent using spibooks.com as an intermediary.

Hype Surrounding Ajax

There is a lot of enthusiastic publicity surrounding Ajax and its effects on the Internet. Several companies that offer complex Web applications, such as MySpace.com, del.icio.us, and Writely.com, have recently been purchased for millions of dollars. Many expensive conferences about Ajax and Web 2.0

Ajax Security Dangers

applications have taken place all over the world. Not only are Ajax applications a hot topic, they are surrounded by an aura of easy money. The hype that Ajax has generated, however, carries its own security ramifications.

First, the hype is driving a massive number of programmers into the Web application field. These novices are using “cut and paste” application development to insert functionality into their projects without a full understanding of the consequences. Most of these programmers are unfamiliar with the security nuances of developing Web applications. This influx of uneducated programmers drives down the average knowledge of those in the field. In turn, more and more technical articles, computer books, and online forums are targeted at novices. Ultimately, there is a small number of experts to provide authoritative answers to an expanding audience.

Second, the demand for these chic, sophisticated Web applications is causing some companies to leap headlong into client-side Web applications. Often a company is simply trying to appear trendy, fend off slick start-up competitors, or provide a new front-end that is fun for their users. Many of these companies do not stop to consider whether their applications should even be Web applications, let alone Ajax Web applications.

Ajax Security Dangers

Recommendations

If you are contemplating converting your business application into a Web application, ask the following questions:

- Should the application be a Web application?
- What is gained 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.
- Ensure that all input is validated before being processed.
- Use white listing rather than black listing for validation. White listing involves accepting what you know to be good data, while black listing uses a list of data not to allow. For example, you know that the zip code should always be five numbers; white listing the zip code input means accepting only five numbers and nothing else.
- Implement the Web application over the Secure Socket Layer (SSL) protocol.

If you decide to retrofit your current Web application or design a new one with Ajax, minimize the program logic that is exposed.

Ajax Security Dangers

Footnotes and References

[1] The actual response time between the client request and the server response only appears to the user to be reduced. The server-side processing time is not affected by Ajax. It is, however, the appearance of a more responsive Web site that is the benefit of Ajax.

[2] For more information about Ajax, refer to the adaptive path essay "[Ajax: A New Approach to Web Applications.](#)"

Ajax Security Dangers

About SPI Labs

SPI Labs is the dedicated application security research and testing team of S.P.I. Dynamics. Composed of some of the industry's top security experts, SPI Labs is specifically focused on researching security vulnerabilities at the Web application layer. The SPI Labs mission is to provide objective research to the security community and give organizations concerned with their security practices a method of detecting, remediating, and preventing attacks upon the Web application layer.

SPI Labs industry leading security expertise is evidenced via continuous support of a combination of assessment methodologies which are used in tandem to produce the most accurate Web application vulnerability assessments available on the market. This direct research is utilized to provide daily updates to S.P.I. Dynamics' suite of security assessment and testing software products. These updates include new intelligent engines capable of dynamically assessing Web applications for security vulnerabilities by crafting highly accurate attacks unique to each application and situation, and daily additions to the world's largest database of more than 5,000 application layer vulnerability detection signatures and agents. SPI Labs engineers comply with the standards proposed by the Internet Engineering Task Force (IETF) for responsible security vulnerability disclosure. Information regarding SPI Labs policies and procedures for disclosure are outlined on the S.P.I. Dynamics Web site at: <http://www.spidynamics.com/spilabs/>.

Ajax Security Dangers

About the Author

Billy Hoffman is a lead security researcher for SPI Dynamics (www.spidynamics.com) where he focuses on automated discovery of Web application vulnerabilities and crawling technologies. Billy is a known industry expert on emerging trends and threats and has been a guest speaker at several industry conferences, and his work has been featured in various industry publications and Web sites. Billy is a reviewer of white papers for the Web Application Security Consortium (WASC), and is a creator of Stripe Snoop, a suite of research tools that captures, modifies, validates, generates, analyzes, and shares data from magstripes. He also spends his time contributing to OSS projects.

About S.P.I. Dynamics Incorporated

Start Secure. Stay Secure.

Security Assurance Throughout the Application Lifecycle.

S.P.I. Dynamics' suite of Web application security products help organizations build and maintain secure Web applications, preventing attacks that would otherwise go undetected by today's traditional corporate Internet security measures. The company's products enable all phases of the software development lifecycle to collaborate in order to build, test and deploy secure Web applications. In addition, the security assurance provided by these products help Fortune 500 companies and organizations in regulated industries — including financial services, health care and government —

Ajax Security Dangers

protect their sensitive data and comply with legal mandates and regulations regarding privacy and information security. Founded in 2000 by security specialists, S.P.I. Dynamics is privately held with headquarters in Atlanta, Georgia. For more information, visit www.spidynamics.com or call (678) 781-4800.

Contact Information

S.P.I. Dynamics
115 Perimeter Center Place
Suite 1100
Atlanta, GA 30346

Telephone: (678) 781-4800
Fax: (678) 781-4850
Email: info@spidynamics.com
Web: www.spidynamics.com