



**FORUM SYSTEMS™**

THE LEADER IN WEB SERVICES & SOA SECURITY

## **AJAX APPLICATION SECURITY AND PERFORMANCE CONSIDERATIONS**

Forum Systems, Inc.

SALT LAKE CITY, UT  
45 West 10000 South, suite 415  
Sandy, UT 84070

TOLL FREE  
1-866-333-0210

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

CONTACT  
Walid Negm  
Vice President,  
Forum Systems Inc.  
[wnegm@forumsys.com](mailto:wnegm@forumsys.com)



## INTRODUCTION INTO AJAX

AJAX (Asynchronous JavaScript and XML) is a cross-platform, client-centric approach for creating highly interactive and responsive Web pages. Ajax is a set of technologies that provide the ability to make asynchronous calls to a web server using an XML based protocol.

The following represents an overview of the four main components that Ajax consists of:

JAVASCRIPT	JavaScript is a general purpose scripting language designed to be embedded inside applications. The JavaScript interpreter in a Web browser allows programmatic interaction.
CASCADING STYLE SHEETS (CSS)	CSS offers a way of defining reusable visual styles for web page elements. In an Ajax application the styling of an application may be modified interactively through CSS.
DOCUMENT OBJECT MODEL (DOM)	The DOM represents the structure of a web page as a set of programmable objects that can be manipulated with the use of JavaScript. Scripting the DOM allows an Ajax application to modify the UI on the fly, effectively redrawing parts of a page.
XMLHTTPREQUEST (XHR)	The XHR object allows an Ajax application to retrieve data from a Web server as a background activity. The data format is typically XML, but works well with any type of text-based data.

JavaScript, CSS and DOM are collectively referred to as DHTML which give developers the features they need to create fairly interactive user interfaces. When the “asynchronous request” was added to this toolset, the longevity of a web page was extended by eliminating the need of the “full page refresh”.

Before the Ajax communication model arrived on the scene, web pages were stateless. Every time a web page needed to change, the entire page had to be posted back to the server for updates. As a consequence user often experienced long waits, especially when large quantities of data are passed between client and server.

With the introduction of this new interaction model, Ajax also provides the necessary tools to create a more interactive and rich user interface. The ability to go back to the server based on an event driven model provides the capabilities compared with those available in a rich UI, such as in Swing desktop applications.

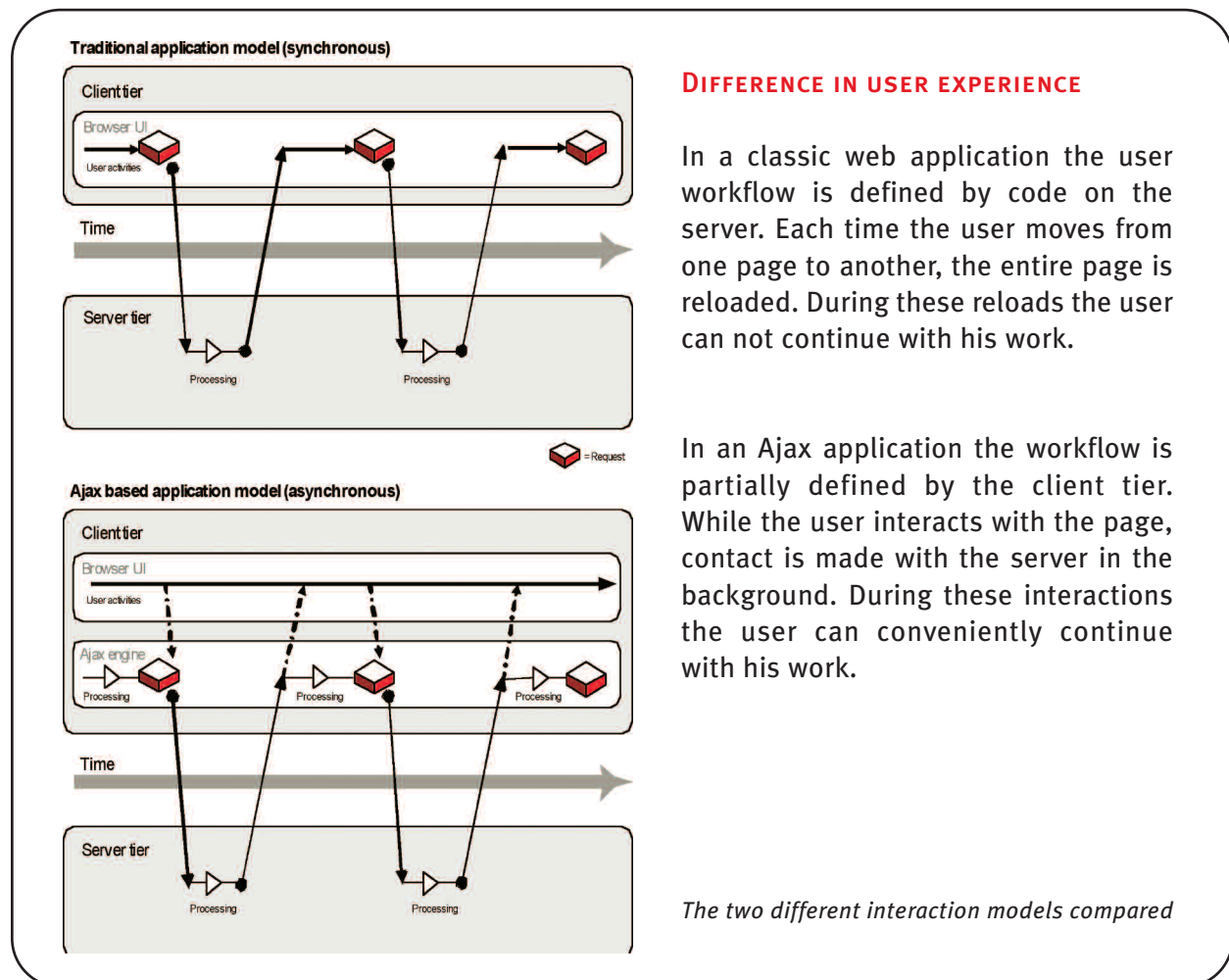


## THE AJAX INTERACTION MODEL

In contrast to the traditional web model, with Ajax there are no more single request and response (pair) restrictions. Instead we have the flexibility to go back to the server when we want. With the introduction of the XMLHttpRequest object, asynchronous processing can be accomplished by starting a HTTP request as a background task.

While the HTTP request is executed in the background, the browser doesn't have to wait for the server to respond. Instead, the client can continue reacting to the user's interaction with the page and deal with the server's response when it eventually arrives.

The asynchronous processing in Ajax is accomplished by registering a callback function with the XMLHttpRequest object. The XMLHttpRequest is dispatched asynchronously, handing control back to the browser. When the server is ready to send back the response, the callback function registered on the XMLHttpRequest is invoked to process the response returned by the server. Finally, the user interface is updated in response to the data from the server, using JavaScript to manipulate the page's HTML DOM.





## WHEN TO USE AJAX

Ajax enables web application developers to apply a different interaction model where they can contact the server in response to user-interaction such as key presses, mouse movements and mouse clicks. With Ajax functionality in web applications we can provide a more flexible and problem-solving approach to the users work.

Typical AJAX use cases are:

- 1. Calculations**  
Calculations such as updating the total amount of a shopping basket or adding shipping cost and sales tax are typically performed on the server. With Ajax there is no need for updating the entire page. All we need to do is send back the (re)calculated amount and update the client.
- 2. Status messages**  
Often an action by the user requires an update of the page. Instead of going back to the server, process the request and resend the entire page with the new information, we could also just update the part that really needs to be updated. Adding a comment to a forum page, updating the number of logged in users or adding a new item to a shopping basket are valid examples where Ajax could be used without having to make a full page reload.
- 3. Populating lists boxes**  
Populating one or multiple list boxes on a user interface can be a challenge in terms of performance and usability. In a traditional web application we have the choice to either send all list box entries in a single burst to the client or populate a list on demand at the cost of making a round trip to the server. With Ajax we have the flexibility to send information to a client as a background task, meaning we can fetch data from the server upon request and populate a list of values when we need it. The most popular implementation is probably Google's search where a list of search results is populated based on the users input as they type.
- 4. Multiple frames**  
Sometimes a web page can be divided into several functional parts. News and information portals are a typical example where different parts of a page have a separate life-cycle. Instead of (re)rendering the entire page when one component needs to be updated, Ajax can provide a way to handle the work of each component individually without one interfering with the other.
- 5. Navigation benefits**  
Before the introduction of the Ajax interaction model we could only handle events from GET requests via links on text and images or from POST requests via submit buttons. In addition, with Ajax we can handle requests from any type user event, such as an onBlur event, a hover or even a key press. Thus being able to respond to a rich event model (in the background), with Ajax we can provide a user interface that is tailored to the actual workflow of the user.



## DIFFERENT WAYS TO APPLY AJAX

The primary data exchange format for Ajax is undoubtedly XML. Even though the XMLHttpRequest Object was designed for XML communication, the Ajax programming model is not tied to a specific data exchange format or programming language. We can contrast:

### DATA CENTRIC APPROACH

The most well known and popular way of fetching data from the server uses the data centric approach. Here we use the XMLHttpRequest to send a request to the server and have the server return an XML response. The callback handler registered with the XMLHttpRequest object uses the responseXML property to get the XML data from the server response. The XML data is then parsed by the client and updated in the DOM.

### CONTENT CENTRIC APPROACH

In a content centric approach, instead of returning XML, HTML content is returned to the client. Another difference is that when not returning XML, we have to use the.responseText property of the XMLHttpRequest object in order to retrieve the response from the server. The HTML returned in the server response is picked up by the callback handler and injected in the DOM via the innerHTML property.

### SCRIPT CENTRIC APPROACH

Applying a script centric approach we return JavaScript code from the server using the.responseText property to fetch the response data. The JavaScript code from the response is then executed by the eval() function on the client. The code contained in the response will have a hook into the client code base in order to make a change or perform an action on client. Typically the client and server side code bases are very tightly coupled using this approach.

## IDENTIFYING VULNERABILITIES

### 1. UN-TRUSTED INPUTS

All HTML based web clients are in a sense open source meaning that anyone can see your client side code, look at the names of form fields and send a request to the server. Whether an application requires authentication or not there will be situations where data is collected and sent to the server in a structure that is unexpected due to:

1. Accidents filling out information
2. Interoperability issues
3. Purposeful corruption of data

An improperly formatted request sent to the server will cause the application to perform unnecessary processing that may cause the application to enter into in-valid or unpredictable states or return exceptions and errors.



Un-trusted input processed inadequately at the server is probably the most prevalent security failure. This is especially true for any Ajax application that uses XML as its primary data format. Due to the verbose nature of XML it's even easier to transport potentially damaging code across a network. Failing to validate all incoming input at the server can have detrimental consequences.

Without adequate security in place this naturally makes your server vulnerable for requests containing corrupted data.

## 2. JAVASCRIPT INJECTIONS

Data that contains illegal JavaScript as part of a data request to or from a server can be a potential risk for security. JavaScript injection attempts are very common and it's always good practice to validate user input in order to make sure it doesn't contain any JavaScript. However JavaScript contained in a request or response is not harmful until some code asks it to be executed. Ajax introduces an additional security risk since a lot of client code is processed by the Ajax client tier using the eval() function. Good application design should make sure that all JavaScript is dealt with accordingly on the server before it can be executed on a client and do damage. On the other hand an Ajax client should also have proper validation in place before it eval(es) any data that's coming from the server. This is even more critical if the client has to deal directly with content or code coming from a third party server.

## 3. SQL INJECTIONS

An AJAX application should not be sending back SQL queries to the server. Instead you should be sending back form values as in strings, numbers that are validated on the server before the SQL statement is executed. Failing to properly inspect and validate data that is used to compose a SQL query can result in unauthorized data access and control of server resources. For example using fixed bind parameters for a database query boosts performance and at the same time is a good practise to prevent unauthorized queries.

## 4. CROSS SITE SCRIPTING

Allowing a user to let code from their own domain interact with scripts from an arbitrary site would open up the potential for a lot of mischief. Effectively, anyone could re-author or deface websites by using DOM manipulation. However the limitations imposed by the JavaScript security model offer protection from this kind of exploit. At the same token this security model also prevents malicious sites from downloading your Ajax client code and pointing it at a different sever without the users knowing that they were talking to a different back-end.

In general we can therefore assume that any code that makes it possible for cross site scripting, is related to a browser vulnerability of some sort. Any Ajax application is limited or rather protected, by the restriction of the browser's security model. However assuming that your application is fully protected by the restriction of this security model alone is not sufficient. Cross site scripting is still one of the biggest security issues as of this day and protecting your data still remains an important issue.

Despite cross site scripting restrictions in place there are legitimate reasons for invoking scripts from domains other than our own. For example, companies may look to provide Web services that are intended to be used by external parties. If we need information from a third party web site, one solution is to make a call to a remote server from your own server and forward it to the client.



Another option to overcome this restriction is by programmatically requesting privileges to perform network activities. This request is eventually passed on to the user in the form of a user-dialog where the user ultimately decides whether or not a resource should be accessed from outside a domain.

#### **5. MAN-IN-THE-MIDDLE ATTACKS**

The web browser that a user is sitting in front of does not enjoy a direct connection to your server. When data is submitted to the server, it is routed across several intermediate points on the internet before it finds your server. It may very well be the case that a malicious router intercepts and reads your data before it reaches its final destination. Ajax uses HTTP both for transmitting the client code and for submitting data requests to the server. As with any web-based application, a malicious entity looking to interfere with your service has several points of leverage. Exploiting these weak points are known as “Man-in-the-middle” attacks.

In order to protect the traffic between an Ajax client and the server, the most obvious measure you can take is to encrypt the traffic using a secure connection. Possible solutions would be HTTPS and MD5. Keep in mind that encryption ensures secure data communication and thus protects you from prying eyes but it doesn't protect you from any attack at the end of the communication channel. You will still need proper security implemented on your server and client or else all you have accomplished is a bad request going through a secured tunnel.

With network encryption, confidentiality is ensured between the point that initiated the encrypted session and the receiver that terminated the encryption protocol. This means that your confidentiality is as strong as the weakest link. If you are looking to ensure data privacy – persistently – end-to-end then message encryption should be considered. This would require some form of client-support for XML Encryption.

#### **6. SCREEN SCRAPING AND RESOURCE THEFT**

A web application is vulnerable to screen-scraping programs that traverse workflows automatically, crafting a request that imitates a form filled in by a user. In the worst case, an automated program can overload a server process using repetitive submissions.

With Ajax applications, data is available as XML and with a few xpath queries (screen scraping) can be accomplished.



## **7. CORRUPTED DATA**

Program data can become corrupted as a result of a program error or an intentional attempt by a malicious attacker. Either way corrupted or malformed data can result in a program disruption that can be detrimental to both server and client. It's essential that your program logic is able to deal with this type of data accordingly. With most Ajax applications using XML as a primary data format it's more detrimental than if the message was in plain text because of parsing, exception handling and data manipulation. Separately, a Browser using resource intensive DOM cannot afford to deal with unruly message structures.

With an Ajax application making extensive use of XML and frequently contacting the server, corrupted XML messages may result in significantly slowing down both client and server. Being able to influence or deliberately affect performance can be a security risk.

## **8. CORRUPTED CODE**

As data can be corrupted, the same can be said about the JavaScript code that an Ajax client receives from a trusted server. An Ajax client in particular is made of JavaScript code that can be dynamically created and modified as a result of a user action. Some Ajax applications even use JavaScript as their primary transport format and send code back to the client which is then eval(ed) on the fly.

In order to prevent client code from being corrupted, the outgoing code can be validated on the server. This could involve verifying the integrity of the code using digital signatures or some form of content integrity verification. This type of security would determine if there were any modifications to the JavaScript (either manually or automatically by a hacker).

## **9. PERFORMANCE**

Good application design should also consider the run-time performance characteristics including throughput, latency, response time and how the application scales under stress and load. With Ajax the number of requests made to the server can increase dramatically and it's important to be aware of how many requests you'll be sending to the server and the resulting server load this will cause. Due to the communication model of an Ajax application, it's also important to deal with issues such as performance and availability when accessing services of third parties. A poorly performing Ajax application can not be considered a security risk by it self, but it can be a major security risk when performance vulnerabilities are located and exploited by others.



## 11. UNAUTHORIZED RESOURCE USAGE

Because XML is clear text any request sent to the server is available to the client for editing. In a simple case a user may determine through trial and error that the “enter order” request structure can be manipulated to format the “update order” request. In this manner they maybe able to:

- Elevate their privileges
- Access unpublished resources
- Execute resources they are not entitled to execute
- Place the application into a invalid or hazardous state

The ability to access resources without the right privileges is a well-known security challenge. The solution is to use access control using identities and credentials such as user-name and password tokens. The difference with Ajax is that the server application must first parse the request in order to determine what resource is being accessed. In an application that is designed with service-orientation in mind, the developer will essentially have a set of services under management and their options are:

- Imperative Security: It is the responsibility of the developer to create and manage the permission of the object that is being invoked.
- Declarative Security: Attributes are used to place security information into metadata in the code and a configuration file holds the permissions.
- Policy-based Security is managed external to the application and policies are used to manage authentication, access control, integrity and privacy.

## 12. DENIAL OF SERVICE ATTACKS

A hacker can leverage an Ajax client to launch an avalanche of requests at the server. They do not need to know anything about the format of the request. They are simply attempting to exhaust the resources of the server be they memory, disk or CPU. Such a malicious attack can be accomplished in a traditional web application, however with Ajax because the requests are verbose this will likely result in resource issues that require detailed and separate attention.

## CONCLUSION

With the arrival of Ajax a new model for building web applications has entered the scene, allowing the developer to create rich user interfaces that respond to events as part of the user workflow. Ajax brings an end to the limitations of the full page refresh cycle. Instead events are handled by the Ajax engine on the client and transmitted to the server for processing while the user can continue with his work.

Even though Ajax doesn't introduce any new major security holes, a poorly designed application may create additional entry points for a malicious entity to detect and exploit vulnerabilities. The responsibility to minimize these entry points to security flaws rests firmly with the server-side programmer. Most security exploits such as Sql injections and XSS attacks are a result of failing to validate incoming traffic on the server. Any request that arrives at the server should be regarded as potentially dangerous and dealt with accordingly.



Ajax also introduces a specific vulnerability owing to the way raw data is provided for consumption from the server. This is particularly the case when an application uses a verbose data format such as XML and needs to deal with services provided by a 3rd party.

## ABOUT FORUM SYSTEMS

Forum Systems, Inc. is the Leader in Web Services and SOA Security™ infrastructure with a comprehensive suite of XML acceleration, trust management and threat protection solutions for the automated Web. Forum Systems' flexible hardware, software and embedded products make vibrant business communications possible by actively protecting XML data and Web services across networks and business boundaries. Forum's products have been chosen by over 150 Fortune 1000 leading enterprises and are winners of numerous industry awards.

The company's products have deep visibility into the XML messages traversing SOA and Web Services deployments using high performance parsing technology. Forum Systems products: Forum XWall, a XML Firewall which has an automatic find feature to discover and guard Web services, Forum Sentry which is a fully featured SOA Gateway, Forum Vulcon, a vulnerability containment service (<http://vulcon.forumsys.com>) and Forum XRay, a Web Services diagnostics system.

Forum Systems has obtained FIPS 140-2 Level II certification and ongoing Evaluation Assurance Level (EAL) Level 4+ validation, a security requirement for many US Government applications.

For more information on how Forum Systems can help you with your Ajax application security and performance considerations, please contact marketing at [info@forumsys.com](mailto:info@forumsys.com)