ICEfaces

Enterprise Ajax Security with ICEfaces

June 2007

Stephen Maryka
CTO
ICEsoft Technologies Inc.

Abstract

The question is simple: Can enterprise application developers deliver Rich Internet Applications using Ajax techniques, but do so in a secure and cost-effective manner?

The evidence is mounting: The Yammer[1] and MySpace[2] worms are two early examples that illustrate Ajax-based implementations are susceptible to attack, and these attacks have the particularly nasty characteristic of being completely invisible to the users being violated, and thus can proliferate at astounding rates.

The solutions are sparse: While the Ajax world is exploding with new capabilities and wiz-bang features, technology providers have been derelict in addressing fundamental security issues in the offerings they promote, leaving a formidable security challenge for the application developer to address.

In this paper we will examine some of the fundamental security issues related to client-centric Ajax techniques, and will show how these issues can be overcome using a server-centric approach based on Java EE and ICEfaces.

The Fundamentals of Ajax Insecurity

The client is untrusted, SO DON"T TRUST IT! This is as fundamental as it gets. The entire web security model is based on the premise of an insecure client, and established and proven security architectures have evolved on this basis. With the popular adoption of Ajax techniques, there is a growing tendency toward client-centric implementations despite the fact that the approach breaks the fundamental premise of the untrusted client. So, regardless of what client-centric Ajax technology you might pick, you will have to address some fundamental security issues.

Client-centric business logic and data: In order to enable a rich user interface, most sophisticated Ajax libraries promote a client run time environment where the user interface and associated business logic and data coexist on the client in order to achieve the rich interactive features required. There is simply no way to protect business logic and data in these types of implementations, which means that the developer must be diligent about identifying sensitive logic and data and determining how best to secure it. It will be necessary to push sensitive parts of the implementation back to the server in order to properly protect them, which can lead to considerable additional complexity in the implementation, and may severely restrict the Ajax features that can be used.

Client-centric validation: While client-side validation using Ajax techniques can be effective for providing immediate feedback to the user, it cannot be trusted to ensure the data submitted back to the server-resident elements of the application is valid and safe. It is necessary to implement a complete server-resident validation subsystem to sufficiently protect server-based assets. Now you face the challenge of maintaining two sets of validation logic, and any inconsistencies between them may open a security hole in the application. Furthermore, an attacker may gain clues on how to attack the application by examining the client-resident validation logic.

XMLHttpRequest (XHR): The XHR is fundamental to all Ajax approaches, and the basic mechanism itself does not introduce additional security concerns, as it inherits the same privileges as the initial HTTP request. Additionally, XHR is not permitted to make cross-domain requests, so in itself XHR can be a secure mechanism. The security issues with XHR arise from its improper/insecure use. One prevalent approach is to use XHR to implement a network interface back into server-resident resources. While this may be the most straightforward

© 2007 ICEsoft Technologies Inc.

Download ICEfaces
http://www.icefaces.org/main/downloads/

way to implement a client-centric, Ajax-based application, the network interface to those protected services cannot be concealed and offers the hacker a well-defined attack surface, if proper access control is not established. The security concerns with XHR are magnified by the invisible nature of mechanism. Any security violation achieved through XHR can occur without being in any way visible to the user, which means that a security breach can propagate rapidly through a community of users that is completely unaware that an attack is in progress. The use of XHR in cross-site scripting attacks can be particularly lethal for this reason.

Implementation Complexity: The distributed nature of a client-centric, Ajax-enabled, application introduces complexity to the application when compared with traditional server-centric approaches. It requires a significant amount of JavaScript development, which in itself introduces complexity to development and testing. Furthermore, the distinct roles of developer, designer, and security expert are blurred in this model. Now your JavaScript developer needs to be a security expert as well. The bottom line is that as complexity rises, the potential for security issues also rises, and the QA process will not typically catch these issues, as they manifest themselves as hidden unintended functionality. This unintended functionality can be fertile ground for the diligent attacker. Beyond the complexity of your own implementation, the implementation of the Ajax framework itself may be inherently insecure. It is a pretty safe bet that a complete security audit has not been performed on your JavaScript framework of choice.

Security Through Obscurity: The notion of the untrusted client trumps the notion that client resources can be protected through some form of obfuscation. The simple fact is, obfuscation provides no real security, and in the worst case may provide a false sense of security to the developer.

Leveraging the Java EE Architecture

From the previous discussion it is clear that there are fundamental security concerns with client-centric approaches, and that extreme diligence will be required to implement a suitable security architecture. There is a big advantage if we can step away from client-centric approaches and back toward server-centric approaches, as we can leverage the existing, industry-proven Java EE security architecture.

An Established Security Architecture: Security has always been a first-class consideration in the Java EE stack and has matured to the point where the security architecture is well established from the persistence layer through to the presentation layer. Furthermore, proven implementations such as JAAS and Acegi are prevalent, so you don't have to invent a security architecture and implement it yourself.

Separation of Roles: Another advantage of the Java EE architecture is that it promotes clean separation of roles between the page designer, Java developer, and security architect. The security architect can establish overall security policies, and identify appropriate technologies for implementation. The Java developer can use the specified security technologies to implement back-end security, and establish necessary access control at various levels of the application architecture. The page designer can deal with high-level concepts like user roles when implementing the UI, and will not have to be concerned about the details of the access control implementation itself.

So where is the Ajax? It is all fine and good to say we are going to use a server-centric approach and leverage the existing Java EE security architecture while maintaining separation of roles during development, but can we achieve effective rich client features based on Ajax techniques with such an approach? We turn our attention to the Java EE presentation layer technologies to make this determination. JSF is the most recent addition to the Java EE stack and it provides a good foundation to build on. JSF itself is not Ajax-enabled and relies on a full-page refresh to affect presentation changes, but it is completely server-centric so provides us with the

Download ICEfaces
http://www.icefaces.org/main/downloads/

security characteristics that we seek. So, if we can establish Ajax functionality in JSF without compromising the server-centric nature of the framework we will be able to inherit the existing Java EE security architecture. This is precisely where the open source ICEfaces technology comes into play.

ICEfaces: Server-centric Ajax with JSF

Preserving the JSF Lifecycle: Central to the JSF architecture is the JSF lifecycle, which is illustrated in Figure 1 below.

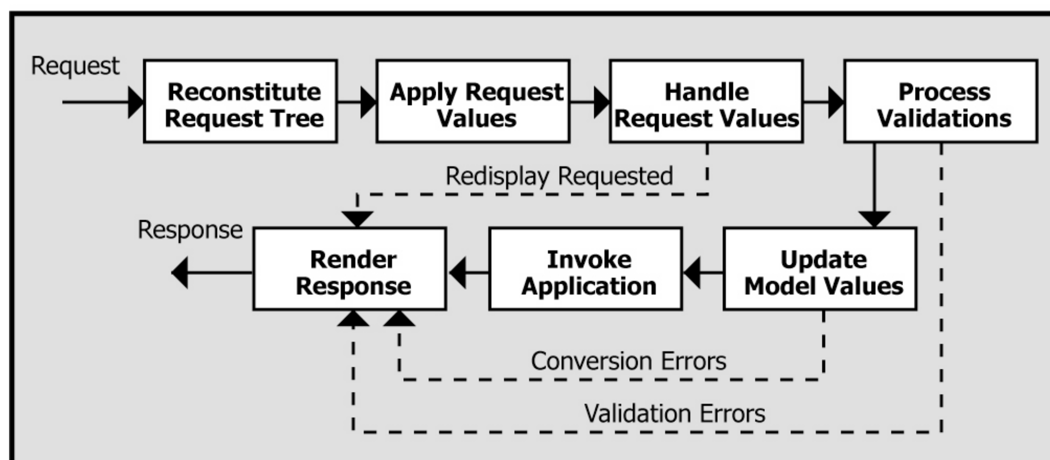

Figure 1: JSF Lifecycle

The lifecycle kicks off with a standard request, which is processed to apply request values, passes through validation, updates the model values, invokes application-specific processing, and finally results in new presentation generated from the render response phase of the lifecycle. If you look to Ajaxify JSF it is important that the approach not circumvent the JSF lifecycle, otherwise you may be circumventing the server-centric security architecture that we seek to preserve. You can expect that if XHR is used from the JSF-generated presentation markup to directly access server-side resources you will be exposed to the same security concerns associated with client-centric approaches. In order to ensure lifecycle preservation, any use of XHR needs to be restricted to standard form submission so the entire JSF lifecycle can execute. There are two key mechanisms required to achieve Ajax functionality under this restriction.

- An automatic, partial submission mechanism is required to react to user interaction with the presentation in order to submit a request back to the server for processing. It is automatic in the sense that each UI control will perform these submissions in a natural way, without requiring developer intervention. For example, when focus moves out of an input text field an automatic submission should occur so that user input can be processed. The submission is partial in the sense that it may not be necessary to process the entire form, as there will likely be various elements in the form that

© 2007 ICEsoft Technologies Inc.

Download ICEfaces
http://www.icefaces.org/main/downloads/

the user has not yet interacted with. For example you don't want validators running against input controls that are still blank.

- An incremental presentation updated mechanism is required to eliminate excessive rerendering of the browser page. Ideally, this mechanism should result in the minimum required updates to the page, and should require no developer intervention. The need for the developer or designer to wire together the elements on the page that may impact each other's presentation under various conditions can be very tedious and error prone. Additionally, the need to do this blurs the separation of roles between the developer and designer.

The ICEfaces Solution: The ICEfaces architecture, as illustrated in Figure 2, provides the required mechanisms to Ajaxify JSF while preserving the lifecycle, and inheriting the underlying Java EE security architecture.

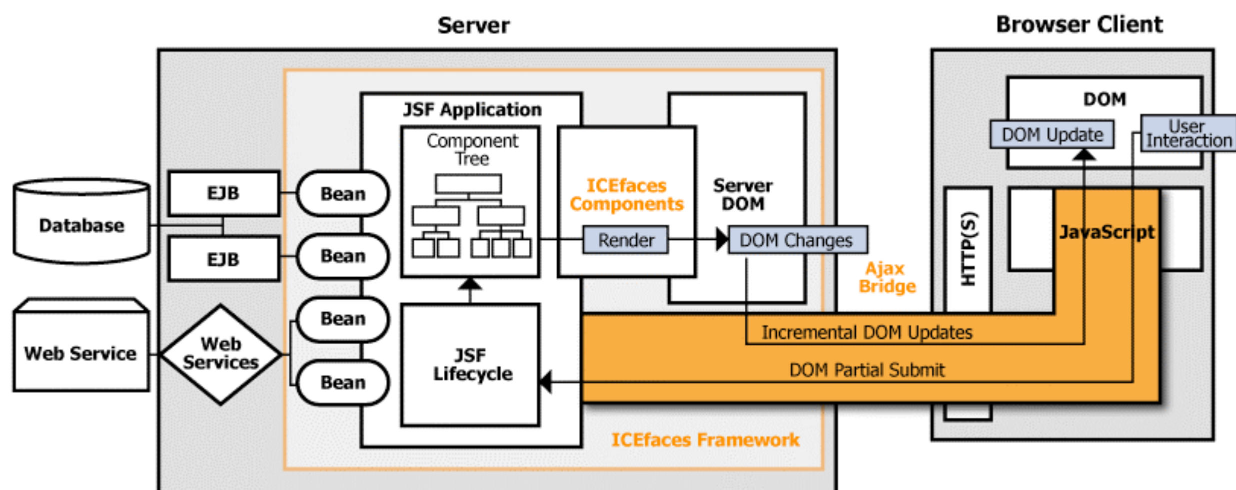

Figure 2: Basic Architecture of ICEfaces

Central to the ICEfaces architecture is a lightweight Ajax Bridge that facilitates both the incremental presentation update mechanism, and automatic partial submission mechanism. While elements of the bridge are implemented in JavaScript and execute client-side, it is a small fixed-function piece of code that exposes only a standard submit mechanism via XHR, and handles only pure presentation data during incremental updates, so does not introduce any security holes to the Java EE security architecture.

The partial submit mechanism is built in to the ICEfaces component suite, so the developer has control over the mechanism on a component level basis, using a standard component attribute. Simply specifying "`partialSubmit=true`" on an ICEfaces component will result in a partial submit when the user interacts with it. The bridge handles the rest of the process in a completely transparent fashion, including limiting the validation process to controls that the user has interacted with.

In order to Ajaxify the render response phase ICEfaces uses a technique called Direct-to-DOM rendering with incremental update. Basically, the entire response is rendered directly into a server-side DOM, and incremental changes to the DOM are distilled out and delivered to the client via the bridge. The client-side of the bridge

© 2007 ICEsoft Technologies Inc.

Download ICEfaces
http://www.icefaces.org/main/downloads/

reassembles the changes in the browser DOM, resulting in smooth incremental page updates. A key advantage to this approach is that the framework determines precisely the minimal updates to be applied, and does so in a transparently fashion. There is no explicit need for the developer to tediously wire together components that might affect each other.

Presentation Layer Access Control: In addition to the fundamentally secure Ajax features of ICEfaces, presentation layer access control is also supported in a natural fashion. Using standard component level attributes with dynamic binds such as a `rendered={#…}`, `disabled={#…}`, `renderedOnUserRole={#…}`, `enabledOnUserRole={#…}` the designer can incorporate role-based access control features into the application UI, and leverage the server-side user authentication mechanisms transparently.

Inherent Security Benefits with ICEfaces

ICEfaces allows Java enterprise developers to build pure, server-centric Java applications that include Ajax functionality without requiring any low-level Ajax development, and that inherit underlying Java EE security characteristics. Concrete security benefits that ICEfaces delivers include the following:

- The application is completely server-centric. No business logic or application data is managed at the client, only pure presentation.
- Validation is all performed server-side, so no mismatches or inconsistencies between client-side and server-side validation can occur.
- XHR is used only for standard form submission. No security holes are opened with XHR acting as a data interface into the server-side application data. The attack surface is limited to the UI itself, eliminating invisible attacks.
- The Ajax Bridge is fixed-size and fixed-function, making it security auditable and testable.
- No dynamic script injection is required, and there are no client-side interpreters that can be compromised.
- JSF automatically escapes output, preventing injection of malicious JavaScript and cross-site scripting attacks.
- Back-end persistence technologies, such as Hibernate, escape SQL input, preventing SQL-injection attacks.
- Existing Java EE access-control techniques can be applied, and presentation layer access control is fully supported in the ICEfaces component suite.
- SSL can be used to secure the connection.

Now if you return to the general security concerns with client-centric Ajax techniques, you will see that the ICEfaces approach addresses them using a server-centric approach. Security is inherent, allowing developers and designers to focus on the creative aspects of application development. So, if you are about to enter the world of enterprise Ajax development and security is paramount, use a server-centric approach based on ICEfaces, JSF and Java EE, and inherit a security architecture rather than inventing one yourself.

© 2007 ICEsoft Technologies Inc.

Download ICEfaces
http://www.icefaces.org/main/downloads/

References

1. Securina (2007). Technical explanation of the JS/Yamanner Worm (2006). Article retrieved June 2007 from: http://secunia.com/virus_information/29782/js.yamanner/

2. Technical explanation of the MySpace Worm (2006). Article retrieved June 2007 from: http://namb.la/popular/tech.html

No part of this publication may be copied in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of ICEsoft Technologies, Inc.

The content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by ICEsoft Technologies, Inc.

ICEsoft Technologies, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this document.

ICEfaces is a trademark of ICEsoft Technologies, Inc.

Sun, Sun Microsystems, the Sun logo, Solaris, Java and The Network is The Computer are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and in other countries.

All other trademarks mentioned herein are the property of their respective owners.

Copyright 2007 ICEsoft Technologies, Inc. All rights reserved.

About ICEfaces

ICEfaces, is an Ajax-based Java EE framework for developing and deploying thin-client, rich enterprise applications.
http://www.icefaces.org

About ICEsoft

ICEsoft Technologies Inc., is the leading provider of standards-compliant, AJAX-based solutions for deploying pure Java, rich Internet applications.
http://www.icefaces.com

ICEsoft Technologies, Inc.

Suite 300, 1717 10th St. NW
Calgary, Alberta, Canada
T2M 4S2

Toll Free: 1-877-263-3822 USA & Canada
Main: +001 (403) 663-3322
Fax: +001 (403) 663-3320

For additional information, please visit: www.icefaces.org

June 2007