

Security Model for the Client-Side Web Application Environments

Sachiko Yoshihama, Naohiko Uramoto, Satoshi Makino, Ai Ishida,
Shinya Kawanaka, and Frederik De Keukelaere

IBM Tokyo Research Laboratory
{sachikoy, uramoto, mak0702, aiishida, shinyak, eb41704}@jp.ibm.com

Introduction

Although best-practice approaches could be effective for securing Web 2.0 applications in the near term, it is time to reconsider the security model of the client-side Web application environment. The current browser security model is designed under an assumption that the content within a server is mutually trustworthy. However, Web 2.0 emphasizes collaboration and interaction of users, which implies that any webpage could include content from multiple participants, including potentially malicious ones. In addition, the use of mashup introduces more chances to integrate potentially malicious content into a single webpage.

Component models, such as the isolation of widgets using `<iframe>` tags, offer effective means for confining content from different sources into its own sandbox. However, they are not a cure-all solution. The current browser model is vulnerable to many attacks, such as cross-site scripting (XSS), as a result of its out-dated security assumptions. Moreover, we cannot force all existing services to follow a new programming model. Since application developers tend to give precedence to presentation over security, a security model that does not change the user experience may be demanded.

To mitigate the risks of attacks in Web 2.0 applications, fine-grained access control in the client-side application environment is important.

Proposed Security Mechanisms

We propose the following three elements as key components of a Web 2.0 security model.

Finer Grained Sandbox Model for the DOM, DOM Events, and Browser Objects

We propose a client-side application sandbox model that can control access to resources at a granularity finer than a browser window. The sandbox model should allow content from different originators (e.g., mashup or wiki entries) to be confined in its own region within a document. The access control should cover the granularity of the DOM objects, DOM events, and non-DOM browser objects such as document cookies and locations.

JavaScript Security

Client-side JavaScript assumes that all content in a window comes from the same source, which is not true in Web 2.0. In order to achieve finer granular access control, we propose an enhanced security model for JavaScript based on the trust domains of the document fragments and stack inspection of the call-path. Though it may seem quite similar to the

Java 2 security model, we take the complexity and flexibility of client-side JavaScript into account, i.e., the event-driven programming model as well as the self-mutating nature of the client-side Web applications. Such security features should be considered as part of the proposed new standards for ECMA Script 4 [1] and JavaScript 2.0 [2].

Another topic is namespace separation [1,2]. The current model does not prevent an attacker from overriding JavaScript variables and functions to alter the application behavior. In addition, the lack of a namespace capability makes it difficult for multiple JavaScript libraries to co-exist in one environment.

Access Control on Networking Capabilities

Asynchronous accesses to remote servers by XMLHttpRequest are also controlled by the same-origin policy. However, the same-origin policy is limited in a sense that it cannot distinguish different paths in the same server. Some browsers allow explicitly relaxing the domain to the super domains, introducing potential vulnerabilities.

In addition, an attacker may interact with a remote server by inserting or modifying a linkable attribute, e.g., a typical XSS attack sends sensitive information such as document cookies to a remote server by using a linkable attribute. Even worse, the attacker may execute arbitrary JavaScript code by inserting a `<script src="..." />` tag, which is executed in the same domain as the main HTML document. By using the `<script>` tag, the attacker may also achieve two-way communications using JSON and callback functions. Since linkable attributes are out of the scope of control of the same-origin policy, there is currently no effective way of preventing such accesses.

We believe that the client-side security model needs to have coherent access control mechanisms for all of the networking capabilities of the browser.

Conclusion

The emergence of Web 2.0 has emphasized the intrinsic weakness of the client-side Web application environment. We have reached a point to retrofit the browser security model. Some of the latest research has already addressed some of the problems by extending browser implementations or by introducing an application-level filters [3,4,5]. However more community collaboration is needed to press for the adoption of such security features as standards.

References

1. June 30, 2003, ECMA Script 4 Netscape Proposal, <http://www.mozilla.org/js/language/es4.html>
2. W. Horwat, JavaScript 2.0, <http://www.mozilla.org/js/language/js20/index.html>
3. P. Vogt et al., Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis, NDSS 2007.
4. E. Kirda et al., Noxes: a client-side solution for mitigating cross-site scripting attacks, ACMSAC 2006.
5. C. Reis et al., BrowserShield: Vulnerability-Driven Filtering of Dynamic HTML, OSDI 2006.