‘Same-Origin Policy’ Circumvention for Legitimate, Dynamic Web Development

Glenn Crawford
BICT student
School of Computing
Christchurch, New Zealand
glencrawford@xtra.co.nz

Malcolm Wieck
School of Computing
CPIT
Christchurch, New Zealand
wieckm@cpit.ac.nz

This quality assured paper appeared at the 1st annual conference of Computing and Information Technology Research and Education New Zealand (CITRENZ2010) incorporating the *23rd Annual Conference of the National Advisory Committee on Computing Qualifications*, Dunedin, New Zealand, July 6-9. Samuel Mann and Michael Verhaart (Eds).

Abstract

The emergence and proliferation of web services and social networking websites has highlighted the requirement to retrieve data from other websites. Without this ability, interactive and engaging web applications that use Asynchronous JavaScript and XML (AJAX)-based applications are hamstrung. Abandoning them would leave today’s users frustrated. However, there are genuine concerns regarding possible misuse of AJAX when employed to access data from different domains. Examples include harvesting sensitive information to effect identity fraud and illegally accessing credit card accounts at the victim’s expense.

The browser-enforced Same-Origin Policy (S-OP) seeks to prevent scripts from facilitating this misuse; the policy enforces considerable limitations to the power AJAX-based applications have when trying to create a richer browser experience for the user, however.

This poster illustrates four different methods that well-intentioned web developers can use to circumvent the S-OP, namely Proxies, HTTP Access Control, Flash/crossdomain.xml and JavaScript Object Notation with Padding (JSONP). Each allows their AJAX-based applications to send requests to a domain other than the requests’ origin, and receive data in response. The

relative merits of the four methods are compared and recommendations are made

Keywords
AJAX, Same-Origin Policy, Web Browser Security, Web-development.

Introduction

Web applications such as Facebook, Windows Live Hotmail, that displays emails instantly when a user selects one from their inbox YouTube, Google Maps, that displays new segments of the map as the user drags their mouse around and Gmail, that automatically checks for new emails, have become part of our lives. Their functionality and lasting appeal relies in part in their ability to operate asynchronously between the client and the web server. Permitting this ability, however, has a downside – it allows malicious exploitation.

Cross-Site Scripting (XSS or JavaScript Injection), is one of many types of malicious attacks that can be deployed against web applications. Exploiting one of a number of known web vulnerabilities, they may allow an attacker to steal users' session cookies or redirect users to malicious Web sites that exploit Web browser security issues. (Cenzic, Inc., 2009, p24)

Content

By 2009, Cross-Site Scripting accounted for 17% of all vulnerabilities found in web applications, (Cenzic, Inc., 2009, p. 11). Use of the S-OP allows scripts running on pages *originating from the same site* to access each other's methods and properties with no specific restrictions.

The ability to create dynamic web content that engages website visitors is significant in creating satisfying user experiences. AJAX technologies potentially offer users such an experience; however the strictures of the Same-Origin Policy make life difficult for the developer. The intention in this paper is to see how best to overcome the hurdles, while respecting the original intentions of the policy. Experiments with the four main known ways of programmatically circumventing the Same-Origin Policy, using both client and server-side web technologies, gave some interesting results. These findings are shown in the poster, along with figures that further explain some of the techniques.

This report will provide example code to demonstrate each circumvention method, and discuss the benefits and drawbacks of each method. In its conclusion, this report will discuss the probable future of making web applications that require cross-site requests, and reflect on which circumvention method is recommended in the meantime.

Conclusion

The Same-Origin Policy circumnavigation methods used each had pros and cons – the one developers should choose depends upon the particular implementation.

References and Citations

Cenzic, Inc. (2009). *Web Application Security Trends Report*. Retrieved March 18, 2010, from http://www.cenzic.com/downloads/Cenzic_AppSecTrends_Q1-Q2-2009.pdf.