Web 2.0 Security Part 3: A MASHup of Problems

Lori MacVittie, 2007-11-07

This is Part 3 of a series on Web 2.0 Security.

A good way to remember things is to use mnemonics, so when you're trying to list the security issues relevant to Web 2.0 just remember this: it's a **MASH**up.

- **M**ore of everything.
- **A**symmetric data formats
- **S**cripting based
- **H**idden URLs and code

This episode is brought to you by the letter "S".

Scripting-based

Web 2.0 technologies, specifically AJAX, are based on the execution of scripts. As we mentioned in Part I of this series, there are a lot **more** scripts than is typically found in a web-based application. While on the server side this is often alleviated by combining multiple scripts into a single application that takes advantage of parameter-based execution that is more closely related to SOA than not, there are also scripts on the **client** that open up new security threats.

In fact, here's a few client-side scripting vulnerabilities that have been discovered - and subsequently exploited:

Yahoo Worm

MySpace Worm

AJAX-Spell HTML Tag Script Injection Vulnerability

These vulnerabilities only scratch the surface of how JavaScript might be exploited. One of the problems with JavaScript is that it's interpreted on the client, and there are no validation mechanisms. That is, malicious JavaScript looks just like valid JavaScript. You can't just examine the script for specific keywords or patterns and determine that the script is malicious.

JavaScript is also self-extensible. That is to say that you can modify existing JavaScript objects - like the XMLHttpRequest object - by forcing the browser to evaluate new JavaScript that extends and adds functionality to the object. And by "forcing" I really mean by delivering a script to the client; the browser will gleefully interpret any script in the page as long as it's in a language it understands.

JavaScript is also dynamic. It can evaluate code that extends itself which in turns evaluates more code and so on. The possibilities are limited only by the creativeness of the author. Where the sandbox (the JVM) was supposed to - and for the most part does - protect the client from most of the really horrible possible exploits such as destruction of your files, it doesn't prevent some of the more subtle exploits dealing with sensitive data such as Cookie Theft or just generally grabbing data from your global clipboard.

The Risks

- There is no way to distinguish malicious script from valid script, leaving attackers free to inject scripts into the client via infected web sites or other techniques that modify the core behavior of Web 2.0 applications
- Developers don't "own" the client (browser) so it's difficult to enforce specific security policies on users that might assist in protecting them from scripting-based vulnerabilities
- Sensitive data can be easily be retreived
- JavaScript is often used to construct URLs for communication; most vulnerability assessment scanners cannot interpret JavaScript and therefore cannot validate the constructed URLs.

> The issue of hidden URLs is the subject of the letter "H", which we'll discuss in the next part of this series.

Next: Hidden URLs

Imbibing: Apple Juice (no, I'm not kidding)

Technorati tags: web 2.0, security, MacVittie, F5, AJAX

Technorati tags: F5, MacVittie, Web 2.0, AJAX, security, application security, Javascript

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com