Understanding Web 2.0

Technologies, Risks, and Best Practices

Companies of all sizes are leveraging Web 2.0 technologies to improve Web site usability and to open new channels of communication. Web 2.0 refers to today's "second generation" of Web technologies, which includes AJAX, RSS feeds, online forums, and mashups. The Web 2.0 term also captures broader development trends, such as:

- Making applications more functionality-rich and responsive
- Generating and sharing content in real time
- Welcoming end user participation

This technical brief examines the underlying technologies used in Web 2.0 applications. It also explains how Web 2.0 introduces some daunting security challenges. New application coding hazards produced by Web 2.0 can elevate the risk of cross-site scripting (XSS) injections, cross-site request forgery (CSRF), unauthorized access, and other Web-based attacks. Lastly, this brief illustrates a number of defense strategies that businesses can use to safely roll out Web 2.0 applications, including application development best practices and dedicated Web application security solutions.

A Brief Look at Web 2.0

Web 2.0 has arrived – both as an industry buzzword and as a real set of technologies that are changing and enriching the Internet experience. With more and more companies adopting Web 2.0, application developers and security professionals should understand the unique security challenges that it poses and outline appropriate defense strategies.

But first: what is Web 2.0?

Web 2.0 is a collection of different Web technologies and design trends. Some of these technologies are new. Many of them have been around for years but only recently have become widely adopted. The three main themes of Web 2.0 are:

- Rich Internet Applications (RIA) – Web applications that mimic traditional thick client applications with persistent connections to the server. Rich Internet Applications rely on AJAX (asynchronous JavaScript and XML) and other protocol extensions to update Web content in real-time without reloading the entire browser window.
- Collaboration – Web sites have morphed from isolated data stores to fully collaborative application platforms. While the early World Wide Web was known for static personal home pages and search engines (as well as painfully slow connection speeds) Web 2.0 is synonymous with user participation and many-to-many communications. Wikis, blogs, social networking sites, online forums, and message boards are all hallmarks of Web 2.0.
- Syndication – In order to broadcast data in real-time and merge content from multiple sources, organizations make a portion of their Web sites available for other sites to use. Common examples of syndication include RSS or Atom feeds and mashups.

While Web 2.0 technologies have extended the capabilities of Web applications and "harnessed the collective intelligence," they have also inadvertently opened up Web sites to a broad range of server and client-side attacks. The companies at the forefront of Web 2.0 technology also demonstrate the hazards associated with it; while organizations like Google, Yahoo, and MySpace have created some of the most interactive and open applications in the world, they also have been plagued with high-profile application vulnerabilities. This brief investigates the vulnerabilities associated with each of the three main Web 2.0 technologies and describes defensive strategies that mitigate these vulnerabilities.

Rich Internet Applications and AJAX

The HTTP protocol is built on the concept of request and response pairs. The client requests a page and the Web server sends it. However, this approach has drawbacks. The Web server cannot send content updates to the client after it has sent the HTTP response. Instead, the server must wait until the client requests the Web page again. Web designers have developed a variety of techniques to build interactive sites with real-time updates without forcing the end user to reload the browser window. These methods include: (1) embedding a thin client such as a Java applet or ActiveX file into the Web application, (2) pushing content updates to the client by not closing the connection and (3) designing the application to pull updates from the Web server.

The most common way to build "Rich Internet Applications" is through the third method, configuring the client application to request content updates from the server. And the easiest way to do this is through client-side code that manually reloads the entire Web page after a specified period of time. Combined with browser frames or iframes, the Web application can update content without reloading the entire browser window. However, this solution is both awkward and inflexible.

In Internet Explorer version 5.0, Microsoft introduced the **XMLHttpRequest** object. Scripting languages such as JavaScript could call the XMLHttpRequest object to send and receive XML data without reloading the browser. It took awhile to catch on, but eventually application developers started adding it to their applications to enhance usability.

In 2005, Jesse James Garrett coined the term "AJAX", or Asynchronous JavaScript and XML, to describe applications with XMLHttpRequests and XML. This term was quickly embraced by the development community as a way to describe the emerging class of Rich Internet Applications. A wide array of applications, including Google Maps, Gmail, Yahoo! Mail, MSN, Flickr, and MySpace, all leverage AJAX. In addition, companies of all sizes are incorporating AJAX into their Web applications. And popular application frameworks like Ruby on Rails as well as packaged business applications now include AJAX.

AJAX Security Threats and Challenges

Unfortunately, Rich Internet Applications like AJAX present new security challenges. There are several reasons why:

- AJAX applications often push much of the application logic to the client. This logic can even include security logic such as access controls and session management. Attackers can adjust client code to easily circumvent policies or masquerade as other users. Attackers can also disable request throttling and other safeguards in order to launch Denial of Service attacks.
- Since more processing is performed by the client, there is a greater risk of exposing server methods to the client. This can enable malicious users to manipulate server methods or to execute restricted commands and methods.
- After an XML request is made, the XML data response is processed directly by JavaScript. In certain cases, the XML data will actually contain script functions to be executed. This makes it much, much easier for attackers to insert malicious code—because character encoding and other security measures are rendered useless. This greatly increases the threat of Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) attacks.
- AJAX increases the opportunity for attack because it incorporates a large number of small modules. Every XMLHttpRequest and response represents a potential target for attack, increasing the overall attack surface.
- A second implication of increased modularity is that application functions are decomposed into multiple smaller elements. This in turn creates issues with state tracking and the validation for modules that work with shared parameters.

- Like other Web 2.0 technologies, the underlying AJAX actions are not visible to end users. It can be difficult for users to see when XML updates occur or identify the source of the XML data. Likewise it is difficult for administrators to identify elements of the application, making the manual configuration of security policy nearly impossible without considerable input from application developers.

Mitigating AJAX Security Threats

There are several actions that all businesses should undertake to secure their AJAX-enabled Web applications. To address the specific vulnerabilities exposed by AJAX, organizations should:

- **Separate data from application code.** Use separate modules for generating display structures and filling in content. Do not incorporate any user input, even if taken from the database, for anything other than body text. For example, never accept or render HTML formatting tags or JavaScript functions supplied by application users.
- **Do not directly execute XML data as script.** For example, do not use the JavaScript ***eval*** method to render XML data.
- **Encode all XML data.** This prevents many types of attacks, from XSS to SQL injection, because dangerous characters like brackets, quotes, and ampersands are interpreted by the browser and the application server as harmless character strings. Use either of the two standard XML encoding methods: numeric character reference (for example, '<' = &#6;) and character entity reference (for example, '<' = &lt;).
- **Never use client side code to perform security related tasks.** Do not expect an attacker to be intimidated by the complexity of your application code. A determined user will always be able to uncover and exploit your application vulnerabilities.
- **Always validate input at the server.** Do not rely on client-side validation checks; these are easy for attackers to circumvent. Input validation can be directly written into application code or enforced with an outside device such as a Web Application Firewall.
- **Always apply access controls and session management at the server.** Like input validation, all access controls and all session management checks should be performed by the server. Otherwise, they can easily be disabled. Again, access control and session management can be performed directly, but also supplemented by an outside device like a Web Application Firewall.
- **When possible, automate the creation and maintenance of security policy.** Because of the large number of small modules and the interaction between modules, the probability of human coding errors is high with AJAX based applications. So providing an external method for supplementing server side security policy is desirable. But because much of the request and response interaction is hidden from the end user, applying external access control and input validation to AJAX applications can be very difficult if a security policy must be set manually. So automating the policy creation and maintenance is a key to successful implementation of external security for AJAX applications.

Collaboration

Over the last several years, Web sites have transitioned from isolated information stores to fully collaborative application platforms. While in the past, most Web sites were closed, with all content produced and managed by the owners of the sites, today many Web sites promote open communications and the freedom for Web users to share ideas and opinions. These trends have produced a new generation of sites that include:

- Wikis
- Blogs
- Social networking sites
- Online forums and message boards
- User-administered Intranet portals

In addition to collaboration-based applications, many traditional Web applications are allowing end user participation on their sites. Examples include Amazon reviews and eBay reputation. Even conventional industries like airlines (Southwest Airlines, www.blogsouthwest.com), banking (Wells Fargo, blog.wellsfargo.com), Automobiles (Ford Motor Company, www.fordboldmoves.com), and Consumer Goods (Clorox, www.drlaundryblog.com) have turned to open blogs to disseminate news, encourage group discussion and build customer loyalty. More and more businesses of all sizes are adding interactive forums to personalize corporate Web sites.

Collaboration – Security Threats and Challenges

While user participation has enhanced Web applications, it has also added new security risks. The most obvious risk is that attackers can post malicious content such as scripts and malware to open forums and blogs. The primary security threats caused by collaborative sites are:

- **Users contributing malicious HTML markup like scripts and page redirects.** If collaborative sites allow HTML formatting tags such as "‹B›" or "‹BR›" or special (non-alphanumeric) characters like quotes, brackets and question marks, they could be exposing their site and their online users to attack.
- **Users uploading malicious files like worms and viruses.** If users are allowed to post GIF and JPEG images, they may also try to upload malware. These types of attacks are primarily targeted at other online users, but they could potentially be used as part of a multi-stage attack on the Web server.
- **Site structure and content is dynamic.** Collaborative sites are constantly changing; every time users post their comments, the site and possibly the hyperlinks, images, and number of URLs change. This dynamic nature hinders traditional security methods like application scanners, manually configured application firewalls and manual reviews of online content.

Mitigating Collaboration Threats

Many of the recommended best practices for securing standard Web applications also apply to collaborative sites. However, with increased attack vectors, many suggested best practices become absolutely essential.

- **Encode all data.** The easiest way to mitigate application threats is to encode data that is submitted by users before it is processed or saved to a database. There may be specific instances where this is not possible, but it is a best practice. All user-supplied data displayed to other users should always be encoded. Encoding user data ensures that this data is interpreted by the client as plain text, not dangerous scripts. There are several different encoding schemes that can achieve this goal: HTML character entity references and numeric character references as well as XML specific encoding methods.
- **Automated user input validation.** Since users are submitting content, it is important to validate all of the data to prevent script injection, file inclusion, command injection and other attacks. As with AJAX security, all of the validation should be performed at the server, not by client side code.
- **Prevent malicious users from posting viruses and malware** – Restrict the types of files that users can upload to known, accepted file formats like GIF images. If the Web application allows any type of file upload, including executables, VBScripts, and other potentially dangerous files, then the uploaded files should be scanned for viruses.
- **Protect packaged applications and frameworks** – Attackers often target vulnerabilities in packaged applications and frameworks because vulnerabilities can be exploited on multiple sites and because attackers can analyze application source code for vulnerabilities (at least for open source frameworks). This is why a large number of vulnerabilities have been discovered in popular online forums, in wiki software, in blogging frameworks and in intranet portal solutions. Web sites that use packaged applications and frameworks should always protect these applications from new vulnerabilities.

Syndication

A key principle of Web 2.0 is ability to share and re-use online data. Syndication allows businesses to publish news and information to an unlimited number of users in real-time. It also enables organizations to combine data from multiple sources. Syndication entails making a portion of a Web site available for other sites to use. The two most common types of Web syndication are RSS feeds and Atom feeds. Mashups, which are another aspect of Web 2.0 related to syndication, make use of syndicated feeds by providing an aggregation of separate feeds into a single feed with added value. For example a real estate agency might provide a mashup that allows users to view home sale prices by location.

All types of organizations today are syndicating online data, from manufacturing companies (Mitsubishi Motors) to financial companies (Fidelity Investments) to casinos (Mandalay Bay). And well known sites like Google Maps, eBay, and Babelfish are pushing the envelope with APIs that allow users to create custom mashups of their content.

Real Simple Syndication (RSS) and Atom

RSS is a popular Web feed format used to publish digital like news feeds, blogs and podcasts. RSS content can be sent directly to users with RSS readers or aggregators. Or RSS feeds can be re-published on other Web sites. There are several different (and incompatible) versions of RSS as well as offshoots like Atom – an XML Web feed format.

Mashups

A mashup is a Web application that combines content from multiple sources together into an integrated service. Named after hip-hop mixes of two or more songs, mashups allow everyday users to combine online search, map, news, photos and other data together to build more useful applications.

Syndication - Security Threats and Challenges

Syndicated data like RSS feeds and mashups present multiple threats.

- **Malicious content in unmoderated feeds can invoke client side vulnerabilities.** Vulnerabilities include buffer overflows and client side execution. This can be due to:
    - RSS readers displaying application content, including injected scripts, literally
    - RSS readers converting encoded data back to the original values (for example, "&lt;" displayed as "<")
- **Increased risk of local zone attacks.** Since many RSS readers translate RSS feeds to HTML and then store the HTML files on the local disk, users are even more exposed to dangerous exploits. This is because the content is now interpreted as part of the local zone. Many versions of Internet Explorer will allow local HTML files to manipulate dangerous ActiveX files that can execute malicious commands.
- **Lack of transparency.** With both mashups and online RSS feeds, it can be difficult for end users to identify the initial source of data. Data may originate from multiple sources and be proxied, parsed, and aggregated before being displayed in the Web browser. This lack of transparency opens up the door for malicious script injections and other client side attacks.
- **Unknown data sources.** Companies that republish content from external sites can inadvertently spread attacks. Since content was developed externally, sites that display online news and content are dependent on the security and integrity of the third parties.

Mitigating Syndication Threats

Organizations that syndicate data or that redistribute content from outside sources can take the following precautions to reduce security threats:

- **Sanitize potentially dangerous content.** Regardless of the data source, if sites do not distribute unknown scripts, files, and HTML markup, they greatly decrease security risk. Tight controls may restrict the ability to format data (for example, italicizing news headlines), so organizations must carefully balance security and functionality requirements.
- **Select upstream content providers carefully.** Companies should evaluate the integrity and the security of external sites before republishing third party data.
- **Stay informed.** Businesses should know the vulnerabilities associated with the various RSS readers. If a content provider is hacked, then sites that republish the content providers' data must either strip the malicious content or find alternative data sources.

Practical Security for Web 2.0

All organizations that plan to roll out Web 2.0 applications should follow best practices for application code development. However, secure code development alone is not enough. Research has shown that even when organizations employ the most painstakingly controlled development processes, it is nearly impossible to write absolutely secure code. Researchers have found that nearly 92% of Web applications are susceptible to some type of attack and 57% are vulnerable to information theft.[1] In addition, Web applications change frequently and new vulnerabilities are often introduced in these updates. Plus, application updates may inadvertently break patches created to fix security holes, re-exposing old vulnerabilities.

Another complicating factor is that organizations may not have access to all application source code – and this inaccessible code may have vulnerabilities. For instance, an organization might use one or more packaged applications like Outlook Web Access and CRM systems or they might have designed their Web sites using a development framework like Ruby-on-Rails, PHP, or .Net. In these situations, even if a company adheres to the strictest code writing processes, its packaged applications and frameworks could expose the company and its online users to attack.

The Threat

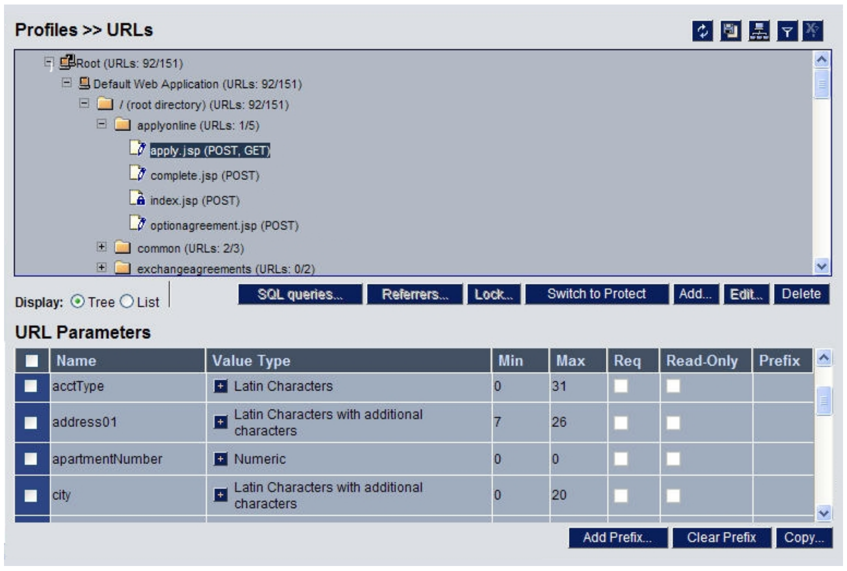
Web-based attacks account for 75% of all Internet attacks,[2] and successful Web attacks can be financially devastating – particularly when sensitive data like customer credit card numbers are at stake. This is why most enterprises rely on a Web application firewall to fortify their Web applications.

SecureSphere Protects AJAX-enabled Applications

Imperva SecureSphere is the first Web application firewall that fully protects Web 2.0 applications, including AJAX-enabled Rich Internet Applications. The SecureSphere Web Application Firewall fortifies Rich Internet Applications with:

- **Integrated Web and XML protection** – SecureSphere protects both HTTP and XML data. Unlike standalone XML firewalls and standalone Web firewalls, SecureSphere can secure both types of traffic simultaneously. In addition, its security architecture is well suited to protect XML content that is constantly changing.
- **Automated application profiling** – SecureSphere dynamically models the structure and elements of protected Web applications. It even automatically determines expected user behavior by inspecting HTTP requests and responses. This is critical for any Web application, but with Rich Internet Applications, automated profiling becomes a necessity. With AJAX specifically, Web pages query a large number of XML files. Besides the increased number of files, XML requests happen "behind the scenes" and are not visible to the end user. Because SecureSphere dynamically profiles applications, customers do not need to manually specify all of the URLs, form fields, cookies, XML files and XML tags. This greatly reduces IT overhead—allowing customers to secure their Web 2.0 applications at a low cost.

[1] "How Safe Is It Out There", Imperva Application Defense Center, 2007.
[2] Gartner, November, 2005



```
Root (URLs: 92/151)
  Default Web Application (URLs: 92/151)
    / (root directory) (URLs: 92/151)
      applyonline (URLs: 1/5)
        apply.jsp (POST, GET)
        complete.jsp (POST)
        index.jsp (POST)
        optionagreement.jsp (POST)
      common (URLs: 2/3)
      exchangeagreements (URLs: 0/2)
```

URL Parameters

	Name	Value Type	Min	Max	Req	Read-Only	Prefix
☐	acctType	Latin Characters	0	31	☐	☐	
☐	address01	Latin Characters with additional characters	7	26	☐	☐	
☐	apartmentNumber	Numeric	0	0	☐	☐	
☐	city	Latin Characters with additional characters	0	20	☐	☐	

SecureSphere automatically builds the Web application profile

- **Server-side security enforcement –** Through its dynamic application profiling, extensive database of attack signatures, HTTP protocol violations, and correlation rules, SecureSphere blocks Web 2.0 attacks that target server side vulnerabilities.
  - **Session tampering –** SecureSphere detects session management exploits like cookie injection, cookie poisoning, session hijacking and session replay.
  - **SQL injection –** SecureSphere blocks attacks aimed at back-end database data.
  - **OS command and file injection –** SecureSphere prevents hackers from injecting malicious commands and content that could compromise a Web server.
  - **Directory traversal –** SecureSphere detects attempts to access files stored outside of the Web application directory on a Web server.
  - **Unauthorized access or manipulation of application modules**
- **Prevention of attacks targeting end users –** Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) are two of the most prevalent AJAX vulnerabilities. Imperva can block all XSS and CSRF attacks that go through a Web server (which means nearly all XSS and CSRF attacks). SecureSphere also prevents other script code injection attacks.
- **Signatures detect specialized attacks –**Signatures block attacks that target specific Web 2.0 application vulnerabilities.

SecureSphere Protects Collaborative Applications

The SecureSphere Web Application Firewall fortifies collaborative applications from online threats. SecureSphere provides the following application defenses:

- **Protection from XSS and CSRF injection –** Client side script injections are the most prevalent attacks aimed at collaborative applications.
- **Protection from a wide range of other application vulnerabilities** - SecureSphere blocks all types of application attacks, from SQL injection and OS command injection to illegal encoding, parameter tampering, buffer overflow, and session management attacks.

- **Protection from Web server software, operating system and network attacks** – SecureSphere protects collaborative applications from all attack vectors from the application level to the network level. SecureSphere has an innovative zero-day Web worm defense that detects a unique combination of attributes that characterize Web worm attacks.
- **Dynamic Profiling** – Using its Dynamic Profiling technology, SecureSphere automatically models the structure and elements of protected Web applications. The profile automatically adapts to application changes. This is a critical requirement when applications are being updated by hundreds or thousands of end users.
- **Protection for packaged applications and frameworks** – SecureSphere applies its dynamic profiling and other security measures to custom and packaged applications alike. However, hackers often target vulnerabilities in application frameworks because vulnerabilities can be exploited on multiple sites and because attackers can analyze application source code for vulnerabilities (at least for open source frameworks). Its attack signatures detect nineteen different online forum vulnerabilities (PHPNuke, Nucleus, Mail2Forum, freeForum, etc), ten Wiki vulnerabilities (TWiki, MediaWiki, PmWiki), four WordPress blog vulnerabilities, and vulnerabilities in other blogging software like A-Blog, CMS, DotClear and JAWS. SecureSphere also detects multiple SharePoint vulnerabilities. Up-to-date application protection is essential for organizations that use packaged Web applications and development frameworks.

SecureSphere Protects Syndicated Data

SecureSphere offers the following security controls for companies that create or redistribute syndicated data:

- **Automated detection of malicious content** – SecureSphere automatically detects Web application attacks, such as XSS and CSRF injections. It also inspects XML files (the format used for RSS 2.0 and Atom feeds) for script tags, illegal characters, and other unsafe content. SecureSphere simultaneously supports white list and black list security models. This flexible architecture offers granular control of security policies; customers can restricted RSS content to “accepted” types of data, or block known, malicious types of data, or combine both security models together to protect Web 2.0 applications without inhibiting functionality.
- **Up to date protection against syndication vulnerabilities** – As with many new technologies, RSS feeds and mashups can have vulnerabilities. These include client (RSS reader) vulnerabilities, protocol vulnerabilities (poorly written APIs for mashups), and compromised content providers. The Imperva ADC delivers up-to-date protection against all of these vulnerabilities.
- **Unified protection for HTTP and XML traffic** – SecureSphere protects Web communications and XML data using the same security technology. Since RSS 2.0, Atom, and many mashup APIs use XML structure for data transfer, it is paramount that any application security solution protects both types of traffic.

Innovate with Web 2.0, Protect with Imperva

Web 2.0 expands application functionality and enriches usability, but it also exposes Web sites to a wider array of attack surfaces. If you plan to deploy Web 2.0 technologies, carefully consider the security implications. First, outline the business objectives of your proposed Web 2.0 applications. Then assess the potential security risks posed by these new applications. Take the appropriate steps to ensure that your Web 2.0 applications are secure: (1) observe application coding best practices and (2) install an application layer firewall to ensure your Web 2.0 applications are secure.

The SecureSphere Web Application Firewall can fortify your Web 2.0 applications from the specific threats posed by Web 2.0. As a standalone appliance, SecureSphere enables security engineers to safeguard Web applications without disrupting the Web development team and without impacting application performance. SecureSphere parses HTTP traffic for all application elements, tracks session information, and records application user activity without needing to terminate HTTP connections. Because of this, SecureSphere supports multiple deployment options – layer 2 bridge, router, proxy, or non-inline sniffer – for drop-in deployment into any network.

Imperva SecureSphere protects many of the world's highest volume Web sites. With its unique Dynamic Profiling technology, SecureSphere delivers accurate and up-to-date application protection without manual configuration or tuning.

For more information about the Imperva SecureSphere Web Application Firewall, please visit http://www.imperva.com/products/securesphere or contact Imperva Sales at sales@imperva.com.

iMPERVA®
Data Security for the Data Center™

US Headquarters
950 Tower Lane
Suite 1550
Foster City, CA 94404
Tel: +1-650-345-9000
Fax: +1-650-345-9004
www.imperva.com

International Headquarters
125 Menachem Begin Street
Tel-Aviv 67010
Israel
Tel: +972-3-684-0100
Fax: +972-3-684-0200