November 2, 2007

RT0764
Multimedia; Security 9 pages

Research Report

Beyond XSS - Towards Universal Content Filtering

Sachiko Yoshihama, Ai Ishida, Naohiko Uramoto

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Beyond XSS - Towards Universal Content Filtering

Sachiko Yoshihama sachikoy@jp.ibm.com
Ai Ishida aiishida@jp.ibm.com
Naohiko Uramoto uramoto@jp.ibm.com

November 2, 2007

1 Introduction

An increasing number of Web applications integrate various content types into single user experience, by using HTML and JavaScript as glue. For example, Flash contents are used by many popular sites not only to show movie contents but also to implement graphical navigation system. Flash players are installed on more than 99% of PCs [1]. PDF has been the most popular digital document format due to availability of the free viewers. There is also a growing need for supporting multimedia contents in user-generated contents; e.g., inserting rich contents, not only static images but also movies and Flash applications, into Weblog postings.

However, these content types, usually handled by the browser plugin, often have their own vulnerabilities. Since each browser plug-ins (e.g., Flash player) per se have full capability of accessing the local resources of PCs (e.g., files or networks), such vulnerabilities may lead to a serious problem that is out-of-control of the Web browsers.

In addition, Contents Feed such as Atom or RSS are also getting popular. Contents feed provide a user with a summary of Web site contents without needing to visit them. However, since data in the content feed will be displayed on Web browsers, it is susceptible to the cross-site scripting attacks. Proactive countermeasures for preventing attacks via content feed are needed in those systems that generate or use content feed.

This document presents a *preliminary* study result of vulnerabilities of various content types, and proposes countermeasures by the content filtering technology. Active Content Filter (ACF) is a filter for HTML, JSON and HTTP requests for filtering out malicious scripting contents. We propose *Universal Content Filter (UCF)*, the next-generation of ACF to support various content types. Although each content type needs further investigation for precisely understanding the security problems, this document hopefully illustrates rough ideas and needs for such study.

2 Flash

A typical Flash file consists of content objects such as movies, images, and sound, tied together with ActionScript which enables programmed behavior of the contents. ActionScript is an extension of ECMAScript, and thus it is compatible with JavaScript which is also based on ECMAScript.

Flash is controlled under the same-origin policy, the de-facto standard security model of Web browsers; i.e., only the Flash and HTML downloaded from the same server can interact each other. However, same-origin policy is enabled only on Flash version 7 or later, and earlier versions have no such limitations. In addition, even in Flash 7 or later, explicit policy relaxation is allowed to enable cross-domain Flash communication.

Because of its flexibility, Flash imposes various threats to Web applications. (More detailed information can be found in [2][3])

2.1 Script Injection to Flash

When request parameters are specified to the URL of a Flash object, the script in Flash can refer to the parameters as global variables in ActionScript. When a Flash application is designed to use

these parameters, it allows a script injection attack by an arbitrary third party.

In fact, some advertisement Flash objects have been designed to take URLs from its request parameters and jump to the URL when the Flash object is clicked. The HTML tag for embedding a Flash object looks as follows:

```
<EMBED src="ad_banner_example.swf?clickTAG=
 http://ad.com/tracking?clickTag=http://www.destURL.com" > ...
```

An attacker can rewrite the URL to specify a piece script code instead of URL.

```
<EMBED src="ad_banner_example.swf?clickTAG=
 http://ad.com/tracking?clickTag=javascript:alert('malicious')" > ...
```

When the Flash application is clicked, it will execute the JavaScript code instead of navigating to the URL.

Flash applications should sanitize the URLs before navigating to them, but many existing Flash applications fail to do so and a protection mechanism that does not rely on the application implementation is needed.

The same threat exists when a Flash application refers to a potentially uninitialized global variable. In such a case, an attacker may add a request parameter with the same name as the global variable, and can successfully inject a script by it.

Note that the attacker does not need to modify a Flash object itself; he can refer to a Flash object from his web page with script injecting request parameters. When the script injection succeeds, the script is executed within the domain from which the Flash object is downloaded; and thus the same-origin policy cannot prevent the malicious script accessing the sensitive resources of the victim's domain that hosts a vulnerable Flash object.

2.2 Malicious Flash Contents

One of the built-in functions of ActionScript is `getURL(url)` by which the Flash application can redirect the user to another Web page. A malicious Flash application may invoke an arbitrary JavaScript code by passing a pseudo JavaScript URL to the `getURL()` function to access sensitive information such as document cookies. E.g.,

```
getURL("javascript:alert(document.cookie)")
```

When a user generated content allows Flash contents in it (e.g., allowing the `<object>` or `<embed>` tag in blog comments), the attacker may carry out a Cross-Site Scripting attack by embedding a Flash object, instead of embedding a `<script>` tag. There are several similar built-in functions (such as `Load` or `loadMovie`) which allows similar attacks.

2.3 HTML Generated by Flash

Flash application can generate HTML elements and inserting it into document.

E.g., ActionScript code which creates an HTML element and insert an `A` element looks as follows:

```
_root.createTextField("tf",0,0,0,800,600);
_root.tf.htmlText = "<A HREF=\"javascript:alert('stolen password: '
  + document.getElementById('password').value );\">Click here</A>";
```

In this example, generated <A> element is executed in the domain of the parent HTML document. That is, even if the Flash object is downloaded from a server different from the parent HTML document, the Flash can access the secret information of the parent document by generating and inserting HTML elements.

Alternatively, `asfunction:` is a pseudo URL protocol which invokes specified ActionScript code. This is similar to the `javascript:` pseudo URLs but it can also refer to functions and variables defined in the Flash application.

```
_root.createTextField("tf",0,0,0,800,600);
_root.tf.text = "<A HREF=\"asfunction:MyFunc,Foo \">Click@Me!</A>";
```

This will increase a chance of attacks through inserted HTML elements.

2.4 Cross-Movie Scripting

Two Flash objects, or a parent HTML and a Flash object, may communicate each other via `GetVariable` and `SetVariable` functions, and thus will introduce a chance of script injection.

In Flash 7 or later, the same-origin restriction has been applied to Flash, and thus communication is limited only to the Flash objects and HTML that are downloaded from the same server. However, in Flash version 6 or earlier, the old rule will be applied; that is, the policy more permissive and allows communication of two objects if they belong to the same super-domain.

Even in Flash 7 or later, the policy can be relaxed by explicitly setting the policy by the `System.security.allowDomain` API.

2.5 crossdomain.xml

The `crossdomain.xml` is a policy file that defines whether the resources on a Web page can be access from Flash applications on different domains. When a Flash application a on the server A access data b on the server B, the `crossdomain.xml` file on B is downloaded and checked by the Flash player to determine whether b can be accessed by a. An example of `crossdomain.xml` is shown below.

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="www.friendOfFoo.com" />
  <allow-access-from domain="*.foo.com" />
  <allow-access-from domain="105.216.0.40" />
</cross-domain-policy
```

A more permissive policy as follows will allow access from any web site.

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy
```

When a web site has too a permissive `crossdomain.xml` policy (e.g., allowing any domains to access the site), it is subject to the CSRF (Cross-Site Request Forgery) attack by using Flash.

CSRF is possible when a Web site authenticates an HTTP request only by a document cookie; any HTTP request that is initiated from a third party web page may be correctly authenticated when the user's browser holds a valid cookie associated with the request URL. A well-known countermeasure is to use a secondary authentication token, e.g., via a `hidden` input field in the HTML document.

A Flash object can access the HTML document body as long as it is downloaded from the same domain, and thus it can retrieve the hidden secondary token from the document. A Flash object can also make arbitrary GET and POST requests to any domains that hosts proper `crossdomain.xml` file which allows access from any domains. Therefore, by using the Flash object, an attacker can carry out CSRF attacks to such servers. (A real-life example of this vulnerability was found in Flickr in 2006 [5] [6]).

2.6 Countermeasures

There are possible countermeasures for threats against Flash.

- Filtering out unauthorized `<object>` and `<embed>` tags in user generated text or HTML contents.
- Filtering out HTML to remove links to Flash that include suspicious request parameters.
- Parsing and analyzing Flash contents to detect suspicious script behavior. (Note static analysis of Flash may be difficult due to flexible nature of ActionScript language; e.g., use of eval() makes it quite difficult to predict the script behavior from static analysis)
- Modify the HTML content and provide proxying functionality to run the Flash object in a separate domain
- Detect vulnerable versions of Flash contents (ver 6 or earlier), or vulnerable `crossdomain.xml` policies to raise warnings or to prevent accesses to such contents.

3 Adobe PDF

Adobe PDF supports JavaScript to allow implementing interactive behavior in documents. Adobe Acrobat's Open Parameter feature [7] allows taking parameters from request URLs, and this allows an attacker to execute JavaScript code by simply referring to a PDF document with the following URL:

```
http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:code_here
```

The attack is reported as Universal PDF XSS [8]. The vulnerability has been fixed in Acrobat Reader version 8 and later; it requests user confirmation before opening a PDF file, but the problem is quite serious if users use older version of the Acrobat Reader.

3.1 Countermeasures

Detection of Universal PDF XSS is similar to detection of attacks to Flash.

4 URL Obfuscation Techniques

Several ways of obfuscating request URL are reported in [8]. Obfuscation prevents users from spotting attacks by themselves (e.g., by looking at the browser's status bar). Detection by the content filter is also more difficult since the suspicious URLs are not present in the contents.

TinyURL. TinyURL (tinyurl.com) is a free Web service that provides short aliases to redirect to long URLs. Attack URL (e.g., URLs with script injecting parameters) can be hidden in TinyURL, and it is more difficult for UCF to detect the attack by scanning contents.

URL rewriting in Apache. You can rewrite request URL to another URL in apache configuration file as follows. As a result, the Web browser will be redirected to the new URL.

```
<IfModule mod_rewrite.c>
 RewriteEngine On
 RewriteRule ^/sample3/(.*)$ http://malicious.com/flash/myflash.swf?x=hehehe [L,R]
</IfModule>
```

Iframes. An attacker may use invisible `iframe` element to open a PDF file in it, e.g.,

```
<iframe
 src="http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:code_hereh
 style=hwidth:0;height:0;border:0></iframe>
```

Note that the document URL can be obfuscated using TinyURL or URL rewriting. An attacker may also load a plain HTML in the `iframe`, and link to the PDF document from the HTML document in the `iframe`. An attacker can stealthily open multiple `iframe` elements in a Web page.

All the tricks listed in this section can be used either for the attacks with Flash or with PDF, or any attacks that may be detected by checking URLs.

4.1 Countermeasures

The URL obfuscation technique may be detected by observing and correlating HTTP request and responses, e.g., by checking HTTP redirecting responses.

5 Feed Injection

Feed injection is a type of the Cross-Site Scripting (XSS) attacks that misuses XML content feed such as RSS and Atom [9].

Content Feed can be subscribed by modern Web browsers, local Feed readers, and Web-based service such as Bloglines. It is dependent on the Feed reader implementation how to interpret the HTML fragments in the body of the content feed. Some vulnerable browsers do not strip special characters (such as < and >) at all, and allows the browser to execute the script tag in the feed.

E.g., below is an example of an RSS feed that includes multiple instances of script injection [9].

```
<?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>
```

Some feed readers are implemented to translate encoded special characters (such as translating < and > into < and >) before interpreting them, and as a result, it allows the browser to execute the script tag in the feed.

In case of local readers, the risk is getting even worse. Some feed readers saves (caches) the feed contents into local files. The saved contents can be later executed with a local file privilege, creating an effect of the Cross-Zone Scripting attack [11].

When feed readers are hosted on a Web site A (e.g., as Bloglines), it is difficult for Web browsers to distinguish the feed contents from the Web site's own content. In addition, the contents on A

might be used by another Web site B that performs as a content aggregator. Each Web site needs to implement its own sanitization mechanism, as well as the means of protection from other vulnerable servers.

5.1 Countermeasures

Content filtering against RSS and Atom is needed. Further requirements need to be investigated.

6 JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a light-weight platform independent data interchange format. JSON is based on the subset of ECMA Script, hence JavaScript. Many Ajax libraries are implemented to convert a JSON string into a JavaScript object with the `eval()` function, because of the performance advantage of using the built-in JavaScript parser. However, JSON messages, especially those include data from third parties or user generated contents, need to be handled with care. Example threats include:

- JSON strings may not be correctly structured as defined in RFC 4267 [12], and include malicious script code that will be executed when the JSON string is evaluated by the `eval()` function. E.g., ``{ x:123, y:'abc' }'')
- JSON strings may include an HTML fragment with malicious script code. E.g., `` { x:123, y:'<script> alert("malicious")</script>' }'' . If the client-side JavaScript code inserts the string into the DOM tree without sanitization, it will result in execution of the malicious script.
- JSON strings may include a malicious URL that will trigger script invocation. E.g., a pseudo URL "javascript:alert('malicious')" may trigger execution of malicious JavaScript code if the URL is inserted into the DOM tree, or accessed by the browser or other plug-in.

6.1 Ajax Library specific JSON implementation

Prototype.js supports an extended HTTP header `X-JSON` that conveys a JSON string in addition to the HTTP response body. The content filter for JSON should take care of such extended headers to detect malicious JSON strings.

Prototype.js also supports an optional security comment delimiters [13] to avoid JavaScript Hijacking attacks [14]. For example, a JSON string

```
{x:123, y: 'abc'}
```

may be encoded by the server with the comment delimiters as follows. Therefore, only a legitimate client-side application, who can access the server via `XMLHttpRequest` (i.e., which is protected by the same-origin policy) can remove the delimiters to retrieve the JSON string in it.

```
/*-secure-\n{x:123, y: 'abc'}\n*/
```

Prototype.js extends JavaScript built-in objects with JSON support, such as the `toJSON()` and the `evalJSON()` function which converts a JavaScript object into a JSON string, and vice versa. The `evalJSON()` function supports an optional `sanitize` parameter; when the `sanitize=true` is specified, the evalJSON() method checks the validity of the JSON string before parsing it with the `eval()` function. However, the use of sanitization is optional and it is disabled by default.

A safe JSON filter needs to understand the implementation differences of various Ajax libraries, to correctly parse and filter out malicious contents from JSON strings. Implementation characteristics of other Ajax libraries need further investigation.

7 XSS Obfuscation

There are various obfuscation techniques to bypass the sanitization algorithms. Those techniques include:

- Using URL encoding, such as encoding ‘‘<’’ into ‘‘%3C’’.
- Using XML entity references, such as encoding ‘‘<’’ into ‘‘<’’.
- Inserting whitespace characters such as a line-brake or a tab space, which are then ignored by some versions of the browsers.
- Using the `data` scheme; when the `data` scheme is specified, the subsequent content is decoded and uesd as the data to be loaded when the URL is activated. The subsequent data can also be encoded by different encoding scheme such as the base64 encoding.

The above encoding tequniques can be used together to make content filtering even more complicated.

E.g., each of the following obfuscated `iframe` element executes JavaScript code `alert('malicious');`.

```
<iframe src="jav ascript:alert('malicious');""></iframe>
<iframe src="jav&#x09;ascript:alert('malicious');"></iframe>
<iframe src="ja
vascript:al
ert('malicious');"></iframe>
<iFrame src
 ="data:text/HTML;base64,amF2YXNjcmlwdDphbGVydChcJ21hbGljaW91c1wnKTs="></iFrame>
<IFRAME src
 ="data:text/html;base64,PHNjcmlwdD5hbGVydChcJ21hbGljaW91c1wnKTwvc2NyaXB0Pg==" />
<IFRAME src
 ="data:text/html;,%3Cscript%3Ealert%28%27malicious%27%29%3B%3C%2Fscript%3E" />
<IFRAME src="data:text/html;,javascript%3Aalert%28%27malicious%27%29%3B" />
```

A safe content filter needs to consider obfuscated XSS. Those filters include not only an HTML filter or a JSON filter but also any content filter that deals with URL representation of active content, such as Flash.

The canonicalization feature that decodes the obfuscated active content in HTML is implemented in ACF V.2 as the C14N filter.

8 Image Format Vulnerabilities

Image files, although they are regarded purely passive content, can be a target of script injection attacks.

8.1 JavaScript Injection to PNG

A vulnerable Web browser executes script injected in a PNG (Portable Network Graphics) file. The script code can be injected into the color pallet (PLTE) field of a standard PNG file. This vulnerability was found on IE6 but fixed by MS07-057 security update.

8.2 PHP Injection to GIF

When PHP code is injected in the Global Color Table field of a GIF file, the file can be mistakenly executed on the Web server as a PHP script. In addition, by leveraging the Remote File Inclusion (RFI) vulnerability of the second victim server, the malicious GIF file on the first server can be imported and executed by the PHP code on the second server.

8.3 Countermeasures

Sanitization of user uploaded image files (e.g., to detect presence of script code in the pallet area) will prevent these kind of attacks. Transformation of image format will also prevent further attacks that belongs to this category.

9 Cross-Site Image Overlaying (XSIO)

An attacker can inject an image into a Web page, and then by using the style sheet, he can layout the image on top of a trusted part of the web page. By image overlaying, an attacker may hide genuine images and instead present untrustworthy information to a user.

In addition, the attacker can associate an arbitrary hyperlink on the image, e.g., to overlay an icon on the header part of a MySpace.com web page, and associate a link to the attacker's Web site. If a user clicks on the image, the user may be navigated to a phishing web site, increasing a chance of successful phishing to the user.

An example attack listed in [10] is as follows:

```
<a href="http://disenchant.ch">
  <img src="evil.gif" style="position:absolute;left:123px;top:123x;" />
</a>
```

9.1 Countermeasures

In-line images with style sheet for absolute positioning can be thought suspicious and may be removed the content filter. However, an attacker may also use relative positioning to carefully layout the image at an arbitrary position. In addition, the layout can be specified either in the style sheet attribute in static HTML, in the `<style>` element, or dynamically specified by JavaScript. More precise analysis of layout would improve detection accuracy

10 Conclusion

This paper presented some possible attack vectors by multi-media content types such as Flash, PDF and images, and proposes evolution of the Active Content Filter (ACF) to the next generation content filter that supports multitude of content types, namely the Universal Content Filter (UCF).

Some attacks presented in this paper can be detected and sanitized by extending the current XML based ACF with additional SAX based filters. Other types of attack requires filtering of multi-media content themselves, such as images and Flash. Because of various URL obfuscation technique as well as attacks that involves multiple victim servers, monitoring of the HTTP protocol will be also needed to detect further attacks.

Finally, although we did not cover in this paper, semantic content filtering is another opportunity for UCF. A semantic content filter may sanitize contents to satisfy compliance requirement such as privacy protection, or to detect and prevent publication of inappropriate social content.

References

[1] Flash Player Penetration, URL: http://www.adobe.com/products/player_census/flashplayer/

[2] Stefano Di Paola, "Testing Flash Applications", 6th OWASP AppSec Conference, Milan, May 2007. http://www.wisec.it/en/Docs/flash_App_testing_Owasp07.pdf

[3] Security Changes in Macromedia Flash Player 7, URL: http://www.adobe.com/devnet/flash/articles/fplayer_security_04.html

[4] Misuse of Macromedia Flash Ads clickTAG Option May Lead to Privacy Breach, 13 Apr 2003. http://www.securiteam.com/securitynews/5XP0B0U9PE.html

[5] Crossdomain.xml security warning, Sep 25, 2006. URL: http://blog.monstuff.com/archives/000302.html

[6] FlashXMLHttpRequest: cross-domain requests, June 04, 2006, URL: http://blog.monstuff.com/archives/000294.html

[7] Adobe Acrobat 7.0 PDF Open Parameters, July 11, 2005, URL: http://partners.adobe.com/public/developer/en/acrobat/PDFOpenParameters.pdf

[8] Universal PDF XSS After Party, Jan 4, 2007, URL: http://www.gnucitizen.org/blog/universal-pdf-xss-after-party/

[9] Robert Auger (SPI Labs), "Feed Injection in Web 2.0", SPI Dynamics, 2006.URL: http://www.spidynamics.com/assets/documents/HackingFeeds.pdf

[10] Sven Vetsch, XSIO - Cross-Site Image Overlaying, Aug 7, 2007, URL: http://www.disenchant.ch/blog/wp-content/uploads/2007/09/xsio.pdf.

[11] Cross-Zone Scripting, Wikipedia, URL: http://en.wikipedia.org/wiki/Cross_Zone_Scripting

[12] D. Crockford, The application/json Media Type for JavaScript Object Notation (JSON), July 2006. URL: http://www.ietf.org/rfc/rfc4627.txt?number=4627

[13] Prototype JavaScript Framework, Introduction to JSON, May 1, 2007. URL: http://www.prototypejs.org/learn/json

[14] Brian Chess, Yekaterina Tsipenyuk O'Neil, Jacob West, JavaScript Hijacking, March 12, 2007. URL: http://www.fortifysoftware.com/servlet/downloads/public/JavaScript_Hijacking.pdf