

FEATURE 3

BLOW UP YOUR VIDEO

Christoph Alme, Dennis Elser
Secure Computing Corporation, Germany

In these days of feature-rich online portals for video and audio files, we are used to seeing interactive multimedia content on websites. Associated file formats are usually perceived by end-users to be trustworthy, with the users expecting them to contain only video and audio content.

Unfortunately it is possible for these file formats to contain more than one might expect. Lately, they have been misused frequently as building blocks in web-centric attack vectors, mostly in combination with cross-site-scripting vulnerabilities as described in [1].

This article presents a round-up of recent multimedia vulnerabilities, looking at today's well known formats for interactive media – *Adobe's Flash* and *Apple's QuickTime* – as well as peeking at *Microsoft's* upcoming web presentation technology, *Silverlight*.

FLASH AND ACTIONSCRIPT

A recent example of a malicious media file [2] demonstrated the transition of a known *Windows* malware behaviour to the *Flash* 'platform'. In a similar manner to a *Windows* executable that detects the presence of a virtual machine and behaves benignly in that environment, the malicious *Flash* file determines whether it is running on a known back-end system that is intended to analyse the malicious impact of *Flash*-based advertising banners, and does not launch its payload if that is the case.

This is possible because *Flash* comes with its own proprietary scripting functionality, a language called ActionScript, and determining the domain that hosts a *Flash* file is as easy as reading an ActionScript property called 'domain'.

Similarly, after injecting a hidden IFRAME element into the hosting website that bears the malicious code, the attack does not run the exploit blindly, but rather checks whether the MS07-009 patch (addressing a vulnerability in an MDAC ActiveX Control) is installed. Only if the check reveals a vulnerable system is the exploit run. The check is as follows:

```
var c = new ActiveXObject
        ("ADODB.Connection");
if (c.Version == "2.7") {
    // ...
}
```

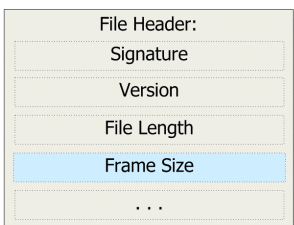
Needless to say, disabling the unspeakable ActiveX functionality (if not using an alternative browser) is advisable.

Rather than going into the details of the plethora of multimedia-related functionality available in ActionScript, we'll focus on one example. As of ActionScript version 2 (from 2004) – which is compliant with the ECMAScript 4 specification – one can invoke any JavaScript function available to the hosting HTML document (e.g. to the *Flash Player* 'container') using an object called 'ExternalInterface'. Wisely, this action is by default (as of player version 8) restricted to media hosted on the same domain as the embedding website. To get the URL of the document embedding a *Flash* file, for example, one can call JavaScript from ActionScript as follows:

```
private var url : String =
    ExternalInterface.call (
        "eval",
        "document.location.href" );
```

This allows powerful interactivity between the JavaScript of a website and an embedded *Flash* object. On the downside, generally speaking it provides another obfuscation layer, allowing malicious JavaScript code to be moved into harder-to-parse *Flash* files.

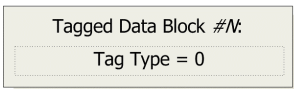
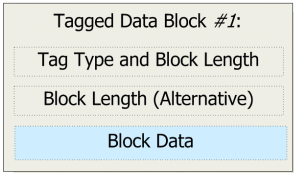
'SWF' (from the former product name *Shockwave Flash*) is a binary file format. It starts with a file header, whose first field contains the magic bytes of either 'FWS' or 'CWS' – the latter identifying the file as being deflate-compressed. Next comes the file format version (one byte), then an unsigned 32-bit field specifying the total file size in bytes, and next the variable-length 'FrameSize' field.



The file header is followed by an arbitrary number of so-called 'Tagged data blocks'. To find the offset of the first data block, one has to calculate the actual length of the FrameSize field, in bits, as:

$$5 + (((\text{FrameSize}[0] \& 0xF8) >> 3) << 2)$$

This must then be translated into the respective number of bytes, rounding up by a byte if the number of bits is not divisible by eight, and adding 12 bytes for the other fixed-size fields of the file header.



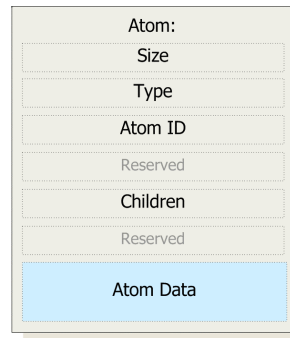
The data blocks each start with a 16-bit field, the upper 10 bits specifying the tag type and the lower six bits the length of the block. If the length is 3Fh, another 32-bit field follows which specifies the block's length. An SWF file ends with a special 'EndTag' block (tag type 0).

Of interest to us is the 'DoAction' block (tag type 0Ch), as it is one of the block types that can contain ActionScript bytecode. Its data is a list of instructions, each called an 'Action' and starting with an eight-bit opcode (called 'ActionCode'). For example, there's an 'ActionGetURL' instruction (opcode 83h) that allows a URL to be opened in a desired target frame – including the current browser window. The length of both the URL and the name of the target frame follows in a 16-bit field behind the instruction opcode, and then comes the URL and the name of the target frame. Both are zero-terminated strings, encoded in UTF-8 (or ANSI in older versions).

QUICKTIME MOVIES AND LINK FILES

QuickTime is another popular format for multimedia, and since it is bundled with *Apple's iTunes* software, it is installed on many end-user PCs. Yet, it's not the scripting-driven, 'interactive' kind of format that *Flash* (for example) is – or is it?

In December last year, a worm based on a *QuickTime* movie file spread by infecting *MySpace* user sites with a link to itself, and using the so-called 'HREF track' to embed JavaScript code into the *QuickTime* movie. URLs (and scripts) in HREF tracks were followed automatically based on elapsed playback time, without user interaction. The JavaScript in turn exploited a cross-site-scripting vulnerability to infect the visiting user's *MySpace* site.



The file format does not start with a file header at a fixed offset, rather the whole file consists of so-called 'atoms'. The presence of an atom called 'movie atom' is mandatory; its type value is 6D6F6F76h ('moov'). Each atom starts with a header, followed by atom-specific data. The size and type fields are each 32-bit, stored in

big-endian order (which is the default in this format). The size refers to the size of the whole atom, including its header, in bytes. If it is set to one, an optional unsigned 64-bit field follows behind the type field that contains the actual size.

An atom can contain other atoms as children, allowing hierarchical storage in movie files. Apart from that, atoms can be stored in (almost) any order. *Apple* recommends certain ordering of atoms, and files that adhere to it can be played while they are being downloaded from the Internet.

About three months after the incident, *Apple* fixed the vulnerability by removing support for embedded JavaScript

– although this was an intended feature, it was probably not a required one.

Just half a year later, something similar seems to apply to scripting ‘capabilities’ discovered in the *QuickTime* player link file. In September [3] this feature proved to allow privilege escalation when combined with *Firefox*’s ‘chrome’ URL protocol.

A *QuickTime* player link file is basically an XML document. It can contain exactly one <embed> element (those other than the first are ignored), which should point to a multimedia file’s URI. This element’s ‘qtnext’ attribute could be tampered with in order to execute JavaScript code with maximum privileges – allowing, for example, the execution of arbitrary executables.

The extension name of these files, which should be ‘.QTL’, seems to be meaningless. Renaming it to ‘.MP3’ and other *QuickTime*-supported file formats is not only possible, but it even removes the last line of defence: *Firefox*’s ‘Open With’ dialog.

Probably in an attempt to make thorough, yet generic detection especially challenging (apologies for the sarcasm), the ‘qtnext’ XML attribute can have many names – up to 256 in fact:

```
<?xml version="1.0">
<?quicktime type="application/x-quicktime-media-link"?>
<embed src="a.mov" autoplay="true"
qtnext3="" qtnext4="" qtnext5=""
qtnext29="... malicious code here ..." />
```

The *Mozilla* team was (once again) quick in providing a *Firefox* update that closed this vulnerability. And with the associated *QuickTime* update released by *Apple* three weeks later, not only had the actual vulnerability been fixed, but the whole scripting ‘feature’ seems to have been removed. Flexibility and susceptibility go together hand in hand.

INTRODUCING SILVERLIGHT

Microsoft’s web presentation platform *Silverlight* [4], which is assumed to be the company’s answer to rival *Adobe*’s *Flash* format, has just been released. While its first version ‘only’ allows the use of JavaScript and VBScript for interactivity within *Silverlight*, the upcoming version (which is already available in Alpha format) will add support for the .NET platform. The available .NET functionality is accommodated to the web browser context though as, for example, classes like ‘System.Web.HttpCookie’ are not accessible from within *Silverlight*.

To make use of *Silverlight*, an embedder has to invoke a script function such as ‘createSilverlight()’ first, passing an ‘Extensible Application Markup Language’ (XAML)

document to be rendered by the plugin. With *Silverlight 1.1*, the document’s ‘Canvas’ element can further include a ‘Class’ attribute in order to reference a .NET managed code assembly that implements event handlers:

```
<Canvas ...
  xmlns:x="http://.../winfx/2006/xaml"
  x:Class="MyNamespace.MyClass;
          assembly=MyAssembly.dll"
  ...>
```

The assembly would implement event handlers, such as for the ‘Loaded’ event (which fires just before the loaded content is rendered):

```
namespace MyNamespace {
  public partial class MyPage : Canvas {
    public void MyPage_Loaded (object o,
  EventArgs e) {
      if (!HtmlPage.DocumentUri.ToString().Contains
  ("ad-verification-domain-here.com")) {
          // ...
      }
    }
  }
}
```

You might think that *Silverlight* is limited to the ‘Windows-with-Internet Explorer’ platform, but hold on: the *Silverlight* browser plugin is already available for *Firefox* as well, and that includes the above-mentioned .NET support. *Mac OS X* with either *Firefox* or *Safari* is supported as well, and support for *Linux* will be realized together with *Novell*, based on the ‘Mono’ project (a cross-platform .NET implementation, that is said to be binary compatible to *Microsoft*’s IL bytecode).

CONCLUSION

The file formats that we have covered briefly here have been shown to be susceptible to cross-site-scripting on a case-by-case basis throughout the last couple of months, and more generally, they allow malicious code to become harder to find. While they cannot be said to be less secure than, say, HTML, the opposite (being more secure) is certainly not the case.

REFERENCES

- [1] Picture theft through hole in Google’s Picasa. <http://www.heise-security.co.uk/news/96554>.
- [2] Yahoo feeds Trojan-laced ads to MySpace and PhotoBucket users. http://www.theregister.co.uk/2007/09/11/yahoo_serves_12million_malware_ads.
- [3] Apple QuickTime Player Zero-Day Vulnerability. http://www.securecomputing.com/SWAT/BlogEntries/SecureBlog_QuicktimePlayer.html.
- [4] Microsoft Silverlight – Light Up the Web. <http://www.microsoft.com/silverlight/>.