AJAX Storage:
A Look at Flash Cookies and Internet Explorer Persistence

Corey Benninger

Foundstone Professional Services & Education
A division of McAfee

The AJAX Storage Dilemna

AJAX (Asynchronous JavaScript and XML) applications are constantly looking for ways to increase their performance. One obvious way to do this is to store more data locally, since data can be loaded from a local file much more quickly than it can be retrieved from a remote website. Imagine an AJAX application pushing down database tables to your browser once and then allowing you to query that data over and over again without going back to the server. This also would work well for applications that allow users to access some functions and data offline, queuing up and saving data until the user can reconnect to the server.

In the past, the data storage solution has been to store data in a cookie, but cookies are limited in size to 4KB per domain. Thus, cookies have generally been used to save small bits of information such as a session ID or perhaps a user's login name for their next visit. Common HTTP cookies can be viewed with a local HTTP proxy and are sent from the website to the web browser and back. There are well-known attack vectors in cookie data–from information leakage to session hijacking to command injection, and more.

Recently, programmers have discovered two technologies that allow for storage greater than the previous 4 KB limits for AJAX applications: Adobe® Macromedia Flash and Internet Explorer's persistence of user data. Using Adobe® Macromedia Flash, an application can save up to 100 KB without user interaction and an unlimited amount, with user agreement. The Dojo AJAX framework already includes features for using this storage container. It is estimated that 95 percent of web browsers have Flash 6.0 or later installed—which is necessary for this feature to work properly. Another storage technology is Microsoft® Internet Explorer's persistence of user data. Applications using this feature can store up to 64 KB per page as an XML file outside of the standard web browser cache.

In this paper, we will explore the use of both of these solutions and explain some of the security implications associated with each solution.

Flash Cookies (Shared Objects)

Flash cookies are not transferred from the client back to the server like HTTP cookies. Instead, downloaded Flash objects that run locally in the web browser read and write these cookie like files. Using JavaScript™, this data can be pulled out of the Flash objects and then used like any other data by the web application. It is not necessary to have any visible signs that a Flash object is running on a given page. For example, the Dojo framework uses a small Flash object that is

downloaded through its JavaScript include files. Functions within the Flash file itself allow for reading and writing the saved data. The Flash file can be renamed and methods around the reading and writing calls can be unique for each application.

It is not necessary to have any visible signs that a Flash object is running on a given page. In fact, it would be difficult to reliably detect if an application were using flash cookies. Using only an HTTP proxy to detect if an application were using Flash cookies would require a way to pattern match on the native Flash function calls on the downloaded Flash binary object (typically a SWF file). This technique would require testing and tools that are not yet available. Currently, the best way to know if a web application is using this technology is to search in the Flash "Shared Objects" data directory. On a Microsoft® Windows® XP computer, this data is typically located in:

```
C:\Documents and Settings\<USERNAME>\Application
Data\Macromedia\Flash Player\#SharedObjects\<RANDOM>\<DOMAIN
NAME>
```

When you drill down in each domain's directory, you will eventually find a "SOL" file. This file contains the data that is stored and used as the Flash cookie.

There are several tools for reading and writing SOL files. Two examples are "*Sol Editor*" (http://sourceforge.net/projects/soleditor), a Windows-based tool and "*SolVE*" by Darron Schall (http://solve.sourceforge.net), which is written in Java™. While string data is typically stored as ASCII text with in SOL files, other object-type data is not as easily viewable. You will almost certainly want to use one of these tools when investigating SOL files during web application assessments. Since SOL files are typically saved under the user's "Documents and Settings" path, they should have the proper ACLs to prevent other users of the same computer from viewing them.

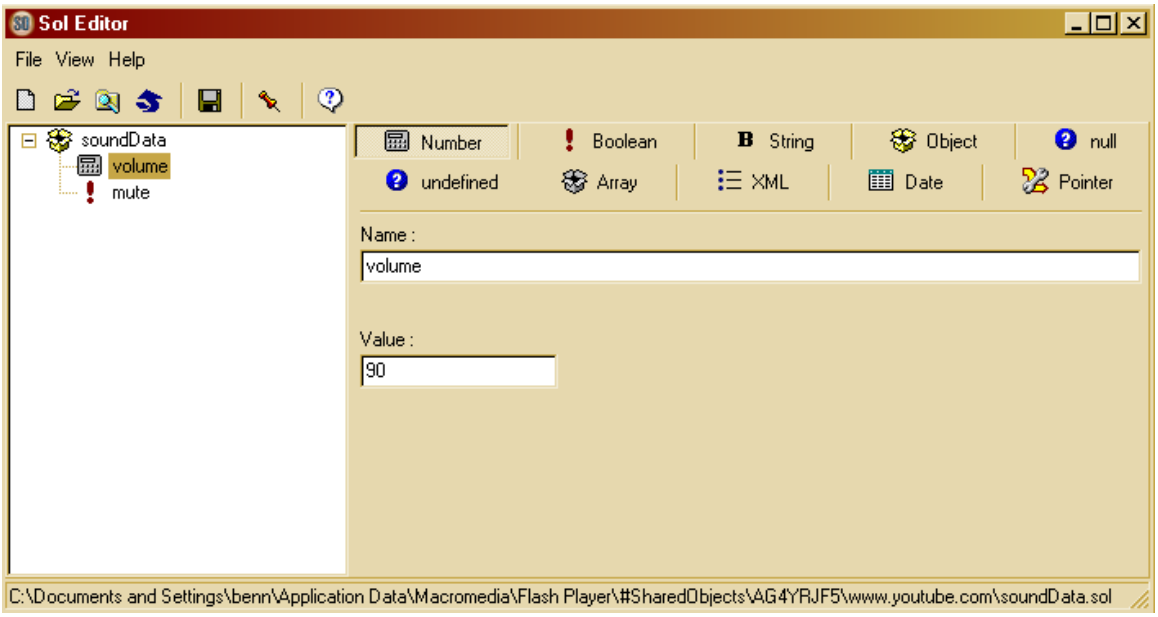

Figure 1: Sol Editor viewing a SOL file from www.YouTube.com

Flash has a default limit of 100 KB per domain for SOL storage. However, a user can set a larger default value, or an application can prompt the user to allow for increased storage when saving data greater than this limit.

Besides the increased storage, Flash cookies offer other advantages over standard HTTP cookies. SOL files are stored outside of the browser's cache, so they are typically are not removed when either Firefox's cache or Internet Explorer's cache is cleared. Another advantage of SOL files is that they are accessible across browsers. A Flash cookie set in Internet Explorer will be available when the site is loaded later in Firefox and vice versa.

You can view which sites have saved Flash shared objects, change the allowed disk size, remove these files, or disable the feature all together. You can do this by editing the XML files in the Flash player's configuration directory. However, if you view certain pages at Macromedia's website, this process is easier when you use the Flash player itself.

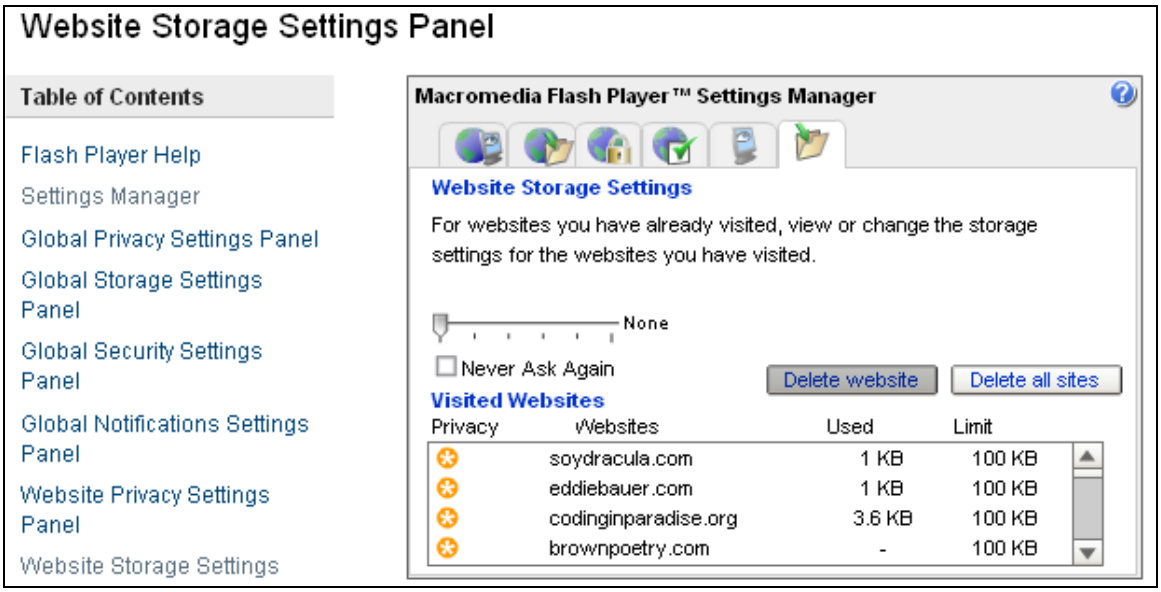

Figure 2: Flash Shared Object settings displayed by browsing to http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager.html

A site can also explicitly allow access to this saved information across domains. This is not the default behavior of a Flash object. A developer would be required to add ActionScript calls within the Flash object itself. One of the required calls would be:

System.security.allowDomain(domain1, …, domainN);

Either explicit domains can be parameters to this function, or the wildcard "*" can be used to allow access from any domain. Once a domain enables this, the application could use the Flash SWF file to set a unique identifier on the user's system. Then any remote web site could include the Flash SWF file and use JavaScript to access its functions and retrieve the identifier. This identifier would be available across various browsers on the end user's system. It would even persist after clearing the standard cache and cookie files from the browser. More information on this feature is available on Macromedia's site:

http://download.macromedia.com/pub/flash/whitepapers/security.pdf

Internet Explorer Persistence

An alternative to Flash cookies is "persistence" which is available only in Internet Explorer web browsers. The "userData" persistence behavior in Internet Explorer allows a web application to write to an XML file up to 64 KB per page (640 KB total per domain). The application has the choice of naming the XML file, but it will reside in a randomly named subdirectory in one of the following two paths.

For Windows XP:
C:\Documents and Settings\<USERNAME>\UserData

For Windows 2000:
C:\Documents and Settings\<USERNAME>\Application Data\Microsoft\Internet Explorer\UserData

When browsing to this directory, most users will notice an XML file there that is used by Windows Update. When the user clicks any of the options in Internet Explorer to clear temporary files, clear cookies, or clear autocomplete data, files in these directories are not removed. Currently, it appears that the only way to remove these files is to delete them manually.

Data values in these XML files are HTML encoded. This helps prevent saved data from corrupting the XML file or being used to escape HTML tags when reloaded. It is possible to manually edit the XML file and modify the values. However illegal characters commonly throw an error message when a page tries to load them. Internet Explorer also uses an index.dat file to limit access to these XML files except to pages within the same directory that originally wrote the data.

If you are looking to implement this feature in web applications, you need to include the following style tag:

```
behavior:url(#default#userdata)
```

Persistence has been a feature of the browser since Internet Explorer 5.0. A CVE (Common Vulnerabilities and Exposures) number was previously assigned to this feature since it was seen as a way to by pass a user's cookie restrictions. Note that only pages within the same directory on the same domain can access previously saved data, so the scope available to use this feature as a type of tracking cookie is limited.

More information about Internet Explorer's persistence behavior can be found at:

http://msdn.microsoft.com/library/default.asp?url=/workshop/author/persistence/overview.asp

Conclusion

Programmers of AJAX applications are continually finding innovative ways of reusing older web technologies. We are likely to see more frameworks and technology that allow increased amounts of data to be saved locally and across domains. This includes plans for future Firefox releases to implement a new client side storage capabilities based on the Web Hypertext Application Technology Working Group (WHATWG) Web Applications 1.0 specifications. Having an understanding of how data is stored and how it can be used is important for understanding the complete picture of the security risks involved.

References:

AJAX MAssive Storage System (AMASS)
http://codinginparadise.org/projects/storage/README.html

Dojo.Storage Test
http://codinginparadise.org/projects/dojo_storage/release/dojo/tests/storage/test_storage.html

CVE-2002-0832: Bypassing cookie restrictions in Internet Explorer
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0832

MSDN Persistence Demo Page
http://msdn.microsoft.com/workshop/samples/author/persistence/load_1.htm

Web Hypertext Application Technology Working Group (WHATWG) Web Applications 1.0 Working Draft – Client Side Session and Persistent Storage
http://www.whatwg.org/specs/web-apps/current-work/#scs-client-side

Mozilla Roadmap Scratchpad (client-local storage improvements)
http://wiki.mozilla.org/Roadmap_Scratchpad

About Foundstone Professional Services

Foundstone Professional Services, a division of McAfee, offers a unique combination of services and education to help organizations continuously and measurably protect the most important assets from the most critical threats. Through a strategic approach to security, Foundstone identifies, recommends, and implements the right balance of technology, people, and process to manage digital risk and leverage security investments more effectively.

Foundstone's Software and Application Security Services (SASS) services help organizations design and engineer secure software. By building in security throughout the Software Development Lifecycle, organizations can significantly reduce their risk of malicious attacks and minimize costly remediation efforts. Services include:

- Source Code Audits
- Software Design and Architecture Reviews
- Threat Modeling
- Web Application Penetration Testing
- Software Security Metrics and Measurement

For more information about Foundstone services, go to www.foundstone.com/services.