

The Scriptless Browser

Simple, Customizable Web Security

Scenera Research LLC

Cary, North Carolina

www.sceneraresearch.com

A browser has been designed to provide a dynamic Web experience without the use of client-side scripts. The browser is further enabled to communicate to servers its willingness to accept cookies, scripts, and other types of files as needed, and provides dynamic behavior through a markup language that indicates where and when dynamic updates can be received from a server. By enabling dynamic Web pages without the use of client-side scripts, and by selectively accepting and refusing Web page cookies and scripts, the scriptless browser eliminates some of the most prevalent sources of malicious software.

Overview

Web browsing is one of the most valuable and ubiquitous ways of gathering information and performing transactions. As the Web expands and evolves, Web browsing becomes more dynamic, with enhanced audio, imaging, text, and other operations. Cookies and scripts help provide those enhancements; however they also make the privacy and security of Web users vulnerable to abuse.

Cookies

Cookies contain small pieces of text that Web servers send to users' hard drives while they browse the Web. Their original name is "tracking cookie" because they help a site remember who its visitors are, recognize when they visit again, and provide helpful information about them when it is needed.

The tiny piece of text inside a cookie is client-side data that a Web application uses to perform any number of tasks. For instance, cookies often hold a Session ID, which is a unique number that the Web site uses to identify a user. When a person accesses a Web site that recognizes her Session ID, the Web site's server uses that ID to look up information about her in its database. This is how the Web site knows her name when she visits and how it automatically fills in her address and other information when she places an order. To gather this data, the Web site server's database collects and stores the information that a user enters when she buys a product online or completes an online registration form, such as her name, e-mail address, postal address, and telephone number.

With the rapid development of the Web in the 1990's, the information that was collected in the cookie databases of Web servers increased in depth and volume, making it more descriptive of users and, therefore, more valuable. Access to the data started being sold to organizations seeking targeted audiences, particularly advertisers. While this may compromise user privacy, it is not to such an extent that many users complained or even noticed. In fact, advertising is accepted by most because it plays an important role in maintaining a free Internet.

While cookies can be valuable, they must be implemented securely to prevent them from being exploited by third parties with malicious intentions. For instance, a Web packet sniffer can intercept a cookie as it passes from a browser to a server and use the cookie to freely access a site session. If this is

done during a session where a user makes an online purchase, the malicious third party may be able to access the user's credit card or steal personal information.

Cookies can also be used with malicious programs that gather and disseminate personal information about users. Typically, this requires a user to download and install a program (usually a "free" program) that installs spyware onto the user's computer along with the cookie. Normally all that happens is that the user receives more spam, postal mail, and telemarketing calls, and the user's computer and Internet connection may slow down as the spyware uses system resources. However, in a worst-case scenario, the user's personal information can end up in the hands of people who attempt to directly contact the individual for nefarious purposes, such as frauds and scams.

Cookie expiration dates help limit the amount of damage that can be done when they are intercepted and most cookies are easy to delete. However there are also flash cookies, also known as Local Shared Objects, which do not expire. Flash cookies can store an unlimited amount of information and cannot be deleted by browsers; however, they can be deleted by conscientious, informed users (3).

In short, cookies can be used in many ways that compromise user privacy and security and must be carefully controlled or eliminated. Web browsers have controls that can eliminate cookies, but Web functions are limited when they are enabled.

Compared to most malware, cookies are insignificant because the great majority is not comprised of computer code or material can be inserted into computer code. A far more dangerous source of Web page malware is the script.

Scripts

Scripts are pieces of computer code that are inserted in the HTML of Web pages and run when those pages are presented in a browser. While they can be powerful, their environment is restrictive and prevents them from operating as freely as a full executable program. However, the Web page environment can have vulnerabilities, known as "holes." When those holes are discovered, nefarious third parties can write malicious scripts that escape the restrictive environment and cause serious harm to Web users and their client devices.

Client-side scripts are the most common type of script to be used maliciously because they are transmitted across the Web onto Web clients, such as computers, smart phones, and PDAs. As mentioned above, scripts are intended to enable dynamic Web page content by updating existing programs and Web devices and broadening the dynamic audio, visual, and interactive experience users have when they visit Web sites. But once they are designed to escape the confines of their Web page presentation environment, scripts can be used to steal users' names and passwords, to set cookies that hijack personal information, or to plant spyware, viruses, worms, and other malicious programs on client devices.

Most browsers today provide add-on tools that block scripts. But, like cookie disabling, script blocking provides limited protection because they cannot be blocked if users want to run Web sites properly. To use the Web to its potential, users are forced to allow scripts to run. Furthermore, when blocking tools are installed, users typically must manually enable scripts to run one at a time, which is tedious, inflexible, and slows Web browsing considerably.

While the most widespread and virulent script malware is controlled by up-to-date antiviral software, many of the cookies and scripts transmitted via Web pages are not controlled by conventional antivirus methods. Furthermore, antivirus protection tools often consume significant resources and can degrade user experiences due to the processing of Web content.

Protection from Cookies and Scripts

As mentioned above, most browsers offer control over cookies and most can also control scripts, but when cookie blocks are enabled or script protection is installed and activated, most Web pages cannot be used for their intended purposes and do not live up to their planned potentials. To have access to all that the Web has to offer, users are forced to enable cookies and allow scripts. And because browsers have no way of distinguishing the good from the bad, the users and their devices remain at risk.

We have designed a browser that can support dynamic Web page performance without the need for scripts, and it protects user and client privacy and security without interfering with Web page operations. It is called the “scriptless browser,” and it is compatible with the Web of today and into the future. We have also designed technology that can be used to eliminate Web page cookies and scripts, completely or selectively, in standard Web browsers and the scriptless browser, now and into the future.

Browsing without Scripts

Client-side scripts are executables that are included in the HTML tags of Web pages and are used to perform dynamic functions, such as automatically closing widgets and presenting dynamic content. But because they are executable, scripts pose risks to users and their client devices.

To minimize risk, we designed simple and straightforward technologies that allow browsers to automatically close Web pages and obtain and enable dynamic content using HTML tags. Because a tag is not a program, it cannot be used to execute malicious software and does not pose privacy or security risks. Nor can tags be used to support interactive dynamic operations locally like scripts do. Tags are simply elements of Web page code that tell browsers how to arrange and display content.

An Example: Automatic Widget Closure Tags

Widgets are resources used in Web media presentation, such as pop-up windows, dialog boxes, buttons, channels, and audio players. A common widget is the one that opens when a user clicks an image, opening a second page with a larger view of the image. The larger image is viewed briefly, and then its page must be closed manually. If the added view pages aren't closed, they can accumulate and create clutter. Today, scripts are often used to clear the clutter.

To automatically close clutter pages without using scripts, the browser must be able to detect that the pages are open and when they need to be closed. An HTML tag is used to open the page, and when each additional page is opened, a condition is set so the browser knows when to close the pages. For instance, the condition can be as simple as a counter that sets a limit on the number of added pages opened. If a limit of five is used, when the sixth page is opened, the first five are closed. A more advanced markup detector can be used, such as one that detects a tag attribute and uses it as a condition for closure. Once the correct condition for closure is detected, the Web pages are closed.

One way to mark the start and end of condition detection is to use new tags we created, <end> and <end/>. The exact conditions for Web page closure are up to the HTML programmer, and can be anything that can be detected by the browser that does not use executable code. The condition may be the presence of another tag, a timer, or any other markup language-compatible non-executable.

Because the automatic closure tags can be used with any type of widget, and because they can be used with virtually any type of detectable condition, this technology has considerable Web potential (1).

Dynamic Web Page Content Tags

Scripts are also used to present dynamic Web page content, such as animation, changing text, and sound. The two most common technologies currently used to access dynamic content are dynamic HTML (DHTML) and asynchronous JavaScript and XML (AJAX). Both of these use scripts to obtain and enable dynamic content, therefore both of them put Web browsers at risk.

To replace scripts in the presentation of dynamic content, we created the dynamic markup language tags, `<dynamic>` and `</dynamic>`, to obtain and display dynamic content in browsers. When a browser parses a Web page and sees a dynamic markup language tag, the tag tells the browser where to find the content to be presented in the same way that it identifies the source of a hyperlink. When it sees the source of the dynamic content, a message is sent to the content's server, and the content is received and inserted in the page in the format indicated by the HTML (2).

Example 1 shows standard HTML code with dynamic markup language tags that identify presentation content to be inserted in a Web page.

```
TABLE-US-00001 EXAMPLE 1 Document URL=pub-sub://helloWorld@unreal.net
<!doctype HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/htmlX/strict.dtd">
<html>
<head>
<title>My first HTML document</TITLE>
</head>
<dynamic source="pub-sub://helloWorld@unreal.us/body">
<body>
<p>Hello world!</p>
</body>
</dynamic>
</html>
```

Example 1. A standard HTML document that uses the dynamic markup language tag to insert dynamic content in a Web page without using scripts.

In this example, the content inside the delimiters of the `<dynamic>` tag is the dynamic content. The source of the content is identified by the URL “pub-sub://helloWorld@unreal.us/body.” Rather than reading and running a script, the HTML tag triggers the retrieval of the information to be inserted into the Web page. Example 1 also uses the publish/subscribe protocol to update the information, which can be valuable in the security of dynamic, real-time Web technology.

Dynamic Tags in Real Time

In the live, real-time Web that is emerging today, dynamic content must be updated faster, more regularly, and automatically. This is important in keeping up with the files and data on real-time Web sites, such as live social networking, online stock and real estate pricing and bidding, and live auctioning.

On Web sites where information or files are continuously updated, the browser will include the dynamic HTML tag plus code that continually requests an update and its location. Instead of a request-response protocol like HTTP, the code will use a publish/subscribe protocol to provide real-time online data updating. When the update runs, the browser will send a request to the update provider, and the updated content that the browser receives will be displayed. This will happen automatically whenever there is an update, and can be set to a specific timing and regularity.

In some situations, continuous content updating must be stopped. This is important when updates that eventually end in fixed content are needed, such as in the editing of written, video, and audio files. To do this, the last version of the dynamic content will be replaced with non-dynamic content and the publish/subscribe protocol will be replaced by HTTP.

For changing from fixed content to continuous updating, the opposite can also be set up, where HTTP is replaced by publish/subscribe to enable content updating. In short, the protocol used for the dynamic portion of the operation does not need to be the same as the protocol that is used in the receiving portion. A publish/subscribe protocol and dynamic tag may be replaced by HTTP or other request-response protocol and applicable tag, and vice-versa.

While scriptless browsing will prevent many serious script infections on the Web, it cannot affect all of them. To further protect Web users and their devices, we also designed a selective blocker that all browsers can use to prevent risk caused by cookies and client-side scripts.

Controlling Cookies and Scripts

To protect users and their client devices from additional risks posed by cookies and client-side scripts, we designed an HTML tag that browsers can use to selectively accept or deny them. This technology can be integrated with the scriptless browser described above, or with current Web browsers. It allows users to set their browsers to accept or block all cookies and all scripts, or to identify specific cookies and scripts to accept and deny.

This browser cookie and script control technology incorporates new content headers in HTTP to accept or reject all cookies or all scripts, no cookies or no scripts, or specific cookies or scripts that are attached to the requested Web page. Because it uses content headers, all blockages and selections are performed before the requested Web page is returned to the browser, so unwanted cookies and scripts are not returned to the browser or the client device.

The content header that controls scripts is called an "Accept-Scripts" header, and it can accept a value of "accepted" or "not_accepted." For backward compatibility with current systems, this header is optional and its absence indicates that scripts are accepted.

The second header is called the "Cookie-Policy" header. It can accept a value of "accepted" or "not_accepted," and is also optional so that it is backward compatible with current systems.

To allow only specific cookies to be accepted, the cookie-policy header is set to "not-accepted" and specific cookies are identified by the user's Session ID in an additional line called "Cookie: sessionid." As described earlier, the Session ID is the piece of data that is stored in cookies to identify the user or device associated with it. It is assigned to a user when she first visits a site and can be used to control the cookie from then on.

Examples of the script and cookie content headers are shown in Example 2. In this example, the Accept-Scripts header has a value of "accept" and the Cookie-Policy header has a value of "not_accepted" with a sessionid cookie identifier that indicates what is accepted (4). The other header lines shown in Figure 1 are standard headers documented in IETF Request For Comments (RFC) 2616 for HTTP version 1.1 (5). Specific scripts can also be controlled by associating them with the user's Session ID in a cookie, as described above.

```
GET www.mySite.us HTTP/1.1
Host: finance.myExample.us.com
```

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.7)
Gecko/20060909 Firefox/1.5.0.7
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pla
in- ;q=0.8,image/png,image/jpeg
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Scripts: accept
Keep-Alive: 300
Connection: keep-alive
Cookie-Policy: not_accepted
Cookie: sessionid=AF13B0C
```

Example 2. An example of a message header that contains the lines that control cookies and scripts. In this instance it uses the Session ID to identify a specific cookie to block.

Values beyond “accept” and “not_accepted” can also be assigned to the cookie and script headers, when needed. For instance, a cookie can be accepted or not accepted based on its user name, password, counter, and other features. A script can be controlled according to its scripting language or other attribute. Control can also indicate that scripts that are accepted cannot access stored cookies and cannot create and store new cookies, providing an additional dimension of security.

For even higher security, scripts can be authorized using an electronic signature. The header can indicate whether a script must be signed and provide a list of authorized signatures to be accepted.

The browser can also operate in a publish/subscribe protocol environment, where the browser that sends the request does not need to wait until it sends an HTTP request to receive a response. This allows the requesting browser to provide an indicator so that the server only sends a response when the status of the indicator warrants it, rather than making the system wait to receive it in response to a request. The flexibility this feature offers can be employed today in the real-time Web environment.

Controlling Other Content Types

Current browsers provide significant control of other content types today. However, Web sites hold many other types of files, such as image, video, and audio files, and those file types are evolving continuously. Any of those files can be infected with malicious software, so we expanded the technology for controlling scripts so that it can be used to control almost any type of file.

In Web pages, the different types of files that are allowed or rejected are set using the “Accept” header. This is an existing header for identifying types of content that are acceptable in a Web page, such as images, videos, and audio files.

Specific content types are defined by their MIME types, which are Internet standards that describe content types. Each MIME type has two elements, a type and a sub-type, separated by a slash (/). For instance, a plain, unformatted e-mail has a type called “text/plain,” and a basic Web page is “text/html.”

Example 2 shows an “Accept” header with the type “text/html,” which is associated with a parameter “charset” that has a value of “us-ascii.” This tells the Web server that receives the page request that a Web page with HTML content (text/html) is allowed, but that its character set must be US-ASCII (6).

```
GET www.mySite.us HTTP/1.1
Host: finance.myExample.us.com
```

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.7)
Gecko/20060909 Firefox/1.5.0.7
Accept: text/xml,application/xml,application/xhtml+xml,text/html;
charset-us-ascii- ,text/plain;q=0.8,image/png,image/jpeg
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: sessionid=AF13B0C
```

Example 3. A message header with an "Accept" header with the MIME type "text/html" associated with parameter "charset" with a value of "us-ascii." This tells the Web server that a Web page with HTML content is permitted, but its character set must be US-ASCII.

Because there is currently no way to provide more detailed information about specific elements associated with a MIME type, we designed two ways to identify specific MIME type elements that will and will not be accepted. The first method specifies the elements that are not accepted, and the second provides a list of the elements that are accepted and determines which ones are not accepted by their omission (6).

Summary

The scriptless browser and cookie and file selection and control functions described here are simple to develop and easy to implement without losing Web-based communication. Once implemented, these technologies do not use undue system resources, and likely use much less than the scripts and blockage tools in use today. Most of all, they have the strength and flexibility needed to operate with current Web browsers, computing devices, and files, as well as those of the future that are evolving today.

Acknowledgment

The author would like to thank Julie Tomlinson for her writing and editing, as well as to recognize her for her tremendous capacity for learning and absorbing new information.

References

1. R.P. Morris. 2008. Methods, systems, and computer program products for providing for automatically closing application widgets based on markup language elements. U.S. Patent Application No. 20080244293. 17pp.
2. R.P. Morris. 2008. Methods, systems, and computer program products for enabling dynamic content in a markup-language-based page using a dynamic markup language element. U.S. Patent Application No. 20080077653. 21pp.
3. How to Manage and Disable Local Shared Objects. 2009. Adobe Macromedia TechNote, <http://kb2.adobe.com/cps/526/52697ee8.html>.

4. R.P. Morris. 2008. Methods and systems for providing for responding without at least one of scripts and cookies to requests based on unsolicited request header indications. U.S. Patent Application No. 20080155013. 15pp.
5. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. 1999. Hypertext Transfer Protocol – HTTP/1.1. Internet Engineering Task Force, RFC 2616:
<http://www.ietf.org/rfc/rfc2616.txt>.
6. R.P. Morris. 2008. Methods and systems for providing for responding to messages without non-accepted elements of accepted MIME types based on specifications in a message header. U.S. Patent Application No. 20080155024. 16pp.

Contact

Contact Scenera Research at info@sceneraresearch.com.