scans port 80, but the `-p` option enables you to scan another port. Remember, just like Stealth, if you use `-p 443`, arirang will scan port 443 but will not handle the SSL connection.

NOTE Netcraft, at *http://www.netcraft.com*, collects Web server information and collates this information based on operating system and Web server software. Its database may seem comprehensive, but it also relies on the target server's header information and response to OS fingerprinting tools such as nmap.

You can also run arirang against a list of Web servers using the `-f` option:

```
$ arirang -G -f hosts.txt
```

Arirang also enables you to scan an IP address range using the `-s` (start) and `-e` (end) options:

```
$ arirang -G -s 192.168.17.2 -e 192.168.17.245
```

The `-P` (uppercase) option helps speed up arirang, especially when specifying the `-f` option or scanning a range of IP addresses. The `-P` option controls the number of processes that can be spawned by arirang. Vulnerability scanning is not a CPU-intensive process, but it does rely on bandwidth and network connections. Multiple processes tend to decrease the amount of time necessary for scans.

```
$ arirang -G -f hosts.txt -P 20
```

Implementation: Creating New Rules

The only useful vulnerability scanner is one that can be manually updated quickly and easily. Arirang includes about 18 scan databases, referred to as *.uxe files. The default location for these databases is determined when you compile the tool. On OpenBSD, the *.uxe files reside in /usr/local/share/arirang.

Single-scan rules are specified with the `-r` option. For example, here's how to check your network for the presence of the Code Red worm or its potential targets:

```
$ arirang -G -s 192.168.0.1 -e 192.168.3.255 \
> -P 20 -r /usr/local/share/arirang/codered.uxe
```

At first, arirang scan rules appear complicated, but most of them follow a simple scheme. Let's take a look at some example rules:

```
200 OK-> HEAD :/index.html.bak^Backup index.html file;Remove backup
 and test files from the web document root;
403-> GET :/admin/^/admin/ directory;;
200 OK-> HEAD :/include/^/include/ directory;Disable directory
 listings;
200 OK-> GET :/msadc/..%255c..%255c..%255c../winnt/system32/cmd.exe?
 /c+dir^IIS Superfluous Decode;MS01-026;
```

An arirang rule is divided into seven fields, although the first field is optional. Table 8-3 describes the components of an arirang rule.

Field	Description
Receive code `[OOB | PEEK | ALL]`	(Optional) Arirang can process a message Out-Of-Band (`OOB`) or Peek into its contents. These are rarely used but are included to make the tool support any type of vulnerability check. The `ALL` is used to wait for a response from the server. Some checks cause a Web server to hang or return no data at all, including headers.
Response code	Normally, this is the numeric response from the Web server. A *404* means not found, and a *200* means OK, which implies that the file exists. Note that arirang requires you to represent a *200* as *200 OK*. Other response codes can be the number—e.g., *403*. This does not have to be the HTTP response code but can be a string of up to 50 bytes (characters) long to search for in the HTML response.
`->`	Delimiter between response code and request method.
HTTP request method	Any HTTP request method defined by the HTTP/1.0 or HTTP/1.1 RFC. `GET`, `HEAD`, and `POST` are used most of the time, but arirang supports techniques such as `TRACE` and `OPTIONS`. The `OPTIONS` method shows what WebDAV capabilities the server supports.
`:<`*`URI`*`>`	The file to check. Arirang supports the URI query string as well—e.g., `login.asp?user=test&pass=test.` Force a rule to apply to a specific port using the syntax *::<port><URL*. For example, `::8080/admin/docs/default.cfg` runs against port 8080. All other options remain the same.
`^<`*`explanation`*`>`	A short description of the vulnerability.
`;<`*`information`*`>;`	A further explanation of the vulnerability, reference to an advisory, or the patch information.

Table 8-3. Arirang "Scan Rule" Format

The explanation and information fields allow you to parse the output into brief or verbose listings. Use the `grep` and `cut` commands to narrow the output.

ALL-PURPOSE TOOLS

The following set of tools are workhorses for making connections over HTTP or HTTPS. Alone, they do not find vulnerabilities or secure a system, but their functionality can be put to use to extend the abilities of a Web vulnerability scanner, peek into SSL traffic, or encrypt a service to protect it from sniffers.

Curl

Where Netcat deserves the bragging rights of super network tool, curl deserves considerable respect as super protocol tool. Curl is a command-line tool that can handle DICT, File, FTP, Gopher, HTTP, HTTPS, LDAP, and Telnet. It also supports HTTP proxies. As this chapter focuses on Web auditing tools, we'll stick to the HTTP and HTTPS protocols.

Implementation

To connect to a Web site, specify the URL on the command line, like so:

```
$ curl https://www.victim.com
```

Automated scripts that spider a Web site or brute-force passwords really exploit the power of curl. Table 8-4 lists some of the most useful of curl's options.

Option	Description
`-H/--header`	Set a client-side header. Use an HTTP header to imitate several types of connections. `User-Agent: Mozilla/4.0` Spoof a particular browser `Referer: http://localhost/admin` Bypass poor authorization that checks the Referer page `Basic Auth: xxxxx` Set a username and password `Host: localhost` Specify virtual hosts
`-b/--cookie` `-c/--cookie-jar`	`-b` uses a file that contains cookies to send to the server. For example, `-b cookie.txt` includes the contents of cookie.txt with all HTTP requests. Cookies can also be specified on the command line in the form of `-b ASPSESSIONID=INEIGNJCNDEECMNPCPOEEMNC;` `-c` uses a file that stores cookies as they are set by the server. For example, `-c cookies.txt` holds every cookie from the server. Cookies are important for bypassing Form-based authentication and spoofing sessions.

Table 8-4. Useful Web-Oriented Curl Options

Option	Description
`-d/--data`	Submit data with a `POST` request. This includes Form data or any other data generated by the Web application. For example, to set the Form field for a login page, use `-d login=arbogoth&passwd=p4ssw0rd`. This option is useful for writing custom brute-force password guessing scripts. The real advantage is that the requests are made with POSTs, which are much harder to craft with a tool such as Netcat.
`-G/--get`	Change a `POST` method so that it uses `GET`. This applies only when you specify the `–d` option.
`-u/--user` `-U/--proxy-user`	Set the username and password used for Basic Authentication or a proxy. To access a site with Basic Authentication, use `-u user:password`. To access a password-protected proxy, use `-U user:password`. This is meaningless if the `–X` option is not set.
`--url`	Set the URL to fetch. This does not have to be specified but helps for clarity when many command-line options are used. For example, `—url https://www.victim.com/admin/menu.php?menu=adduser`. Curl gains speed optimizations when multiple URLs are specified on the command line because it tries to make persistent connections. This means that all requests will be made over the original connection instead of establishing a new connection for each request.
`-x/--proxy`	Set an HTTP proxy. For example, `-x http://intraweb:80/`.
`-K/--config`	Set a configuration file that includes subsequent command-line options. For example, `-K www.victim.com.curl`. This is useful when it becomes necessary to specify multiple command-line options.

Table 8-4. Useful Web-Oriented Curl Options *(continued)*

Case Study: Password Guessing

So far we've delineated a few of the useful options that curl offers, but it still doesn't really seem to do much of anything. Curl's power, however, lies in its adaptability to any Web (or other protocol) situation. It simplifies making scripts. Perl, Python, and C have libraries that aid HTTP connections and URL manipulation, but they require many support libraries and a steeper learning curve. That is not to say that Perl can't do anything curl can do—curl is just easier. It's one reinvention of the wheel that raises the bar for other tools.

The following script demonstrates how to use curl as a customized brute-force password guessing tool for a Web site. The Web site uses Form-based authentication in a POST request. The login process is further complicated by a cookie value that must be passed to the server when the user logs in and is modified if the password is correct.

```
#!/bin/sh
# brute_script.sh
# Use curl and a password file to guess passwords in form-based
# authentication.  2002 M. Shema
if [ -z $1 ]; then
    echo -e "\n\tUsage: $0 <password file>"
    exit 1;
fi
PASSLIST=`/bin/cat $1`
USERNAME=administrator
# change the COOKIE as necessary
COOKIE="MC1=V=3&LV=20013&HASH=17C9&GUID=4A4FC917B47F4D6996A7357D96;"
CMD="/usr/bin/curl \
  -b $COOKIE \
  -d user=$USERNAME \
  -c cookies.txt \
  --url http://localhost/admin/login.php"
for PASS in $PASSLIST; do
  # specify Headers on this line to work around inclusion of spaces
  `$CMD \
    -H 'User-Agent: Mozilla/4.0' \
```

Password Guessing *(continued)*

```
    -H 'Host: localhost' \
    -d passwd=$PASS`
  # upon a successful login, the site changes the user's cookie value,
  # but we don't know what the new value is
  RES=`grep -v $COOKIE cookies.txt`
  if [ -n '$RES' ]; then
    echo -e "found $RES with $USER : $PASS\n";
    exit 0;
  fi
done
```

We find a dictionary of common passwords and then run the script against the target. If we're lucky, we'll find the administrator's password. If not, we'll move on to the next user.

OpenSSL

Any Web attack that can be performed over port 80 can also be performed over port 443, the default SSL port. Most tools, exploit code, and scripts target port 80 to avoid the overhead of programming encryption routines and handling certificates. An OpenSSL proxy enables you to redirect normal HTTP traffic through an SSL connection to the target server.

Implementation

The OpenSSL binary is more accurately a suite of functionality, most of which we will not use. If you were to type **openssl** on the command line without arguments, you would be sent to the openssl pseudo-shell:

```
$ openssl
OpenSSL>
```

Obviously, OpenSSL contains more functionality than we need to set up a proxy. We are interested in the SSL/TLS client, or the `s_client` option. You cannot obtain usage information by typing **s_client –h**, but it does have a man page. Now we can connect directly to an SSL server using the `s_client` command. The `–quiet` option reduces the amount of error information:

```
$ openssl s_client –quiet –connect www.victim.com:443
depth=0 /C=fr/ST=idf/L=paris/Email=webmaster@victim.com
verify error:num=18:self-signed certificate
verify return:1
```

```
depth=0 /C=fr/ST=idf/L=paris/Email=webmaster@victim.com
verify error:num=18:self-signed certificate
verify return:1
HEAD / HTTP/1.0
Date: Tue, 26 Feb 2002 05:44:54 GMT
Server: Apache/1.3.19 (Unix)
Content-Length: 2187
Connection: close
Content-Type: text/html
```

When we type **HEAD / HTTP/1.0**, the server returned its header information, thus confirming that the SSL connections succeed. The lines previous to the `HEAD` command indicate the certificate's information and status. It includes the distinguished name (DN, for you LDAP enthusiasts) and the e-mail address of the person who created the certificate. OpenSSL also indicated that the certificate was self-signed—that is, it has not been verified or generated under a third-party certificate authority (CA). For the most part, we ignore these errors as long as we can establish the SSL connection.

NOTE In a true e-commerce situation, the validity of a server certificate is extremely important. The certificate's domain should always match the domain of the URL that it protects, it should not be on a revocation list, and it should not be expired.

Now we could save some typing by piping the `HEAD` request into the `s_client` command:

```
$ echo -e "HEAD / HTTP/1.0\n\n" | \
> openssl s_client -quiet -connect www.victim.com:443
```

This puts us one step closer to being able to make raw requests of an HTTPS server, but it doesn't solve the problem of using a tool such as arirang to scan an SSL server. To do so, we need to run the `s_client` command in a proxy situation. In the previous examples, `s_client` connected to the SSL server, an HTTP request was sent, an HTTP response was received, and then the connection closed. Arirang or Stealth could make more than 6000 requests. Obviously, we need a better degree of automation.

The Unix (and Cygwin) inetd program solves this problem. The inetd daemon runs on a system and listens on specific TCP and UDP ports. When another host requests to connect to one of the ports that inetd monitors, inetd makes a quick access check and then passes on valid connection requests to another daemon. For example, most Unix FTP servers operate from the inetd daemon. A file called /etc/inetd.conf contains an entry that instructs inetd how to handle FTP requests:

```
# /etc/inetd.conf example content
ftp    stream    tcp    nowait    root   /usr/libexec/ftpd   ftp -US
```

The first column, `ftp` in this case, represents the port number on which the service listens. The value *ftp* could be replaced with *21*, the default FTP port, and everything would

still function properly. How does this help us set up an SSL proxy? Well, we just create a new service that listens on a TCP port of our choice. Then, instead of launching an FTP daemon, we launch our `s_client` command:

```
# /etc/inetd.conf SSL proxy example content
80    stream    tcp    nowait    root    /home/istari/ssl_proxy.sh
```

The /home/istari/ssl_proxy.sh file contains two lines:

```
#!/bin/sh
openssl s_client -quiet -connect www.victim.com:443 2> /dev/null
```

NOTE Setting up an SSL proxy on an Internet-facing server might have unexpected consequences. Always restrict access to the SSL proxy using the /etc/hosts.allow and /etc/hosts.deny files, or their equivalents for your Unix variant.

Now whenever a connection is made to the localhost on port 80, the connection is forwarded over SSL to *www.victim.com* on port 443. Any connection that you wish to make to the victim server is made to the localhost (or the IP address of the proxy) instead. This runs all of arirang's Unix scan rules against *https://www.victim.com*:

```
$ arirang –G –h localhost –p80 –r unix.uxe
```

The principle drawback of this technique is that the scans must always target a single host. The SSL proxy is not a true proxy in the sense that it performs protocol translation from arbitrary hosts to arbitrary hosts (a many-to-many configuration). Instead, it accepts requests from any host to a specific host (a many-to-one configuration). Consequently, you cannot scan IP address ranges through an SSL proxy, but at least you can still test a server that has only the HTTPS service running. Of course, if you use whisker, you don't need to worry!

Case Study: Inted Alternative

Inetd is not the only method of launching a service. It does have the advantage of being able to apply TCPWrappers, a method for allowing or denying access to a port based on IP address. Not all operating systems use inetd, and the Windows operating system definitely does not have this function.

Cygwin If your friends still pick on you because you're running some version of Windows, don't fret. The Cygwin environment has an inetd daemon and the OpenSSL software that allows you to run an SSL proxy. Cygwin does complain about using *80* for the service name. The /etc/inetd.conf file should contain the following:

```
# /etc/inetd.conf Cygwin SSL proxy example
www    stream  tcp   nowait  root  /home/ssl_proxy.sh ssl_proxy.sh
```

Inted Alternative *(continued)*

Then you can run inetd from the command line. We like to run it with `-d`, the debugging option, just to make sure everything works correctly:

```
$ /usr/sbin/inetd.exe -d /etc/inetd.conf
```

Now the proxy is listening on port 80 and forwarding connections to the target specified in the ssl_proxy.sh script.

Installing inetd as a native Windows service takes a few more manipulations. There are two methods of creating the service. The prerequisite for each is that the Windows PATH environment variable contains C:\cygwin\bin or wherever the cygwin\bin directory resides. Inetd can install itself as a service:

```
$ /usr/sbin/inetd.exe --install-as-service /etc/inetd.conf
```

To remove it, use the `--remove-as-service` option.

Cygwin's built-in utilities also install and run the inetd service:

```
cygrunsrv -I inetd -d "CYGWIN inetd" -p /usr/sbin/inetd -a -d
 -e CYGWIN=ntsec

cygrunsrv -S inetd
```

The `-R` option removes the inetd service.

Xinetd Xinetd puts a little "extra" into the inetd daemon. It improves logging, connection handling, and administration. On systems that support xinetd, the service definitions are usually in the /etc/xinetd.d directory. Create an SSL proxy service using this xinetd syntax:

```
#default: off
#description: OpenSSL s_client proxy to www.victim.com
service 80
{
    socket_type = stream
    wait = no
    protocol = tcp
    user = root
    server = /root/ssl_proxy.sh
    only_from = 127.0.0.1
    disable = no
}
```

Inted Alternative *(continued)*

As always, be aware of running services with root privileges and services to which only you should have access.

Netcat (sort of) For one-off connections, such as running a compiled exploit that normally works against port 80, Netcat saves the day. You may not be able to run a whisker scan correctly, but a single connection will succeed. Whisker has the advantage of working on Unix and Windows systems, provided the OpenSSL suite is installed. A Netcat pseudo-proxy fits in a single command:

```
$ nc -vv -L –p 80 -e "openssl s_client -quiet \
> -connect www.victim.com:443"
```

The `–L` option ("listen harder") instructs Netcat to continue listening even if a client closes the connection. The `–e` option contains the `s_client` command to connect to the target. Then, connect to port 80 on the listening host to access the SSL server on the target (*www.victim.com* in the example).

You will have to use the original version of Netcat to do this. On OpenBSD, for example, the `–L` option is replaced by `–k` and the `–e` option is deprecated since Unix supports pipes (`|`).

An OpenBSD command looks like this:

```
$ nc –vv –k –l 80 | openssl s_client –quiet \
> –connect www.victim.com:443
```

Of course, it doesn't make sense to add the extra step of using Netcat. You should be able to pipe the output of the exploit directly into the `s_client` command, skipping a step. Then again, there may be scenarios in which strict network controls or mixed OS environments actually make this useful.

Stunnel

OpenSSL is excellent for one-way SSL conversions. Unfortunately, you can run into situations in which the client sends out HTTPS connections and cannot be downgraded to HTTP. In these cases, you need a tool that can either decrypt SSL or sit between the client and server and watch traffic in clear text. Stunnel provides this functionality.

You can also use stunnel to wrap SSL around any network service. For example, you could set up stunnel to manage connections to an Internet Message Access Protocol (IMAP) service to provide encrypted access to e-mail (you would also need stunnel to manage the client side as well).

Implementation

SSL communications rely on certificates. The first thing you need is a valid PEM file that contains encryption keys to use for the communications. Stunnel comes with a default file called stunnel.pem, but you can make your own with the `openssl` command:

```
$ openssl req -new -out custom.pem -keyout custom.pem -nodes -x509 \
> -days 365
...follow prompts...
$ openssl dhparam 512 >> custom.pem
```

Now the custom.pem file is ready for use. Stunnel looks for stunnel.pem by default, or you can use your own with the `-p` option.

Cygwin Compile Note You will need to edit the stunnel.c file to compile stunnel on Cygwin. Comment the following lines, which appear on or around line 391:

```
/*        if(setgroups(1, gr_list)) {
            sockerror("setgroups");
            exit(1);
          } */
```

The byproduct is that you cannot use the `-g` option to specify alternative group privileges when you run stunnel, but you are not likely to be in a scenario in which this is demanded.

Monkey in the Middle What if you need to view the data being sent over an SSL connection? You might need to examine the data passed between a Web-based client application and its server, but the client transmits in HTTPS and the server accepts only HTTPS. In this case, you need to slip stunnel between the client and server, downgrade the connection to HTTP so it is readable, and then turn the traffic back into HTTPS so the server accepts it. This requires two stunnel commands.

Run stunnel in normal daemon mode (`-d`). This mode accepts SSL traffic and outputs traffic in clear text. The `-f` option forces stunnel to remain in the foreground. This is useful for watching connection information and making sure the program is working. Stunnel is not an end-point program. In other words, you need to specify a port on which the program listens (`-d <port>`) and a host and port to which traffic is forwarded (`-r <host:port>`). The following command listens for SSL traffic on port 443 and forwards non-SSL traffic to port 80. If we're just making a monkey in the middle, the `-r` points to the other stunnel command:

```
$ stunnel -p custom.pem -f -d 443 -r <host>:80
2002.04.15 16:56:16 LOG5[464:1916]: Using '80' as tcpwrapper service
 name
```

```
2002.04.15 16:56:16 LOG5[464:1916]: stunnel 3.22 on
 x86-pc-mingw32-gnu WIN32 with OpenSSL
0.9.6c 21 dec 2001
2002.04.15 16:56:16 LOG5[464:1916]: FD_SETSIZE=4096, file ulimit=-1
 (unlimited) -> 2000 clients allowed
```

The other stunnel command is similar, but it is used in client mode (`-c`) to accept traffic in clear text and output traffic encrypted by SSL. In this example, the command listens on port 80 and then sends SSL traffic to the final destination on port 443:

```
$ stunnel –p custom.pem –f –d 80 –r www.victim.com:443 –c
2002.04.15 17:00:10 LOG5[1916:1416]: Using '80' as tcpwrapper service
 name
2002.04.15 17:00:10 LOG5[1916:1416]: stunnel 3.22 on
 x86-pc-mingw32-gnu WIN32 with OpenSSL
 0.9.6c 21 dec 2001
2002.04.15 17:00:10 LOG5[1916:1416]: FD_SETSIZE=4096, file ulimit=-1
 (unlimited) -> 2000 clients allowed
```

If we run these commands on different computers (or between a computer and a VMware session), we can sniff the traffic that is forwarded over port 80.

SSL for a Service Stunnel provides the same functionality of inetd with the addition of SSL encryption. Stunnel supports TCPWrappers natively, which means that it checks the /etc/hosts.allow and /etc/hosts.deny files upon starting. This makes it possible for you to apply encryption to just about any service. For example, IMAP is a protocol for remote mailbox access. The drawback with IMAP is that passwords can be sniffed.

This is what the IMAP service configuration looks like when run from / etc/ inetd.conf:

```
imap     stream  tcp     nowait  root    /usr/sbin/tcpd imapd
```

The service name is imap (TCP port 143); the TCPWrappers daemon executes the IMAP daemon.

Now take a look at the equivalent service configuration under stunnel. The following command would be run from the command line, not as part of /etc/inetd.conf:

```
# stunnel -p imapd.pem -d 143 -l /usr/sbin/imapd.exe  -N imapd
2002.04.15 17:08:38 LOG5[1820:1680]: Using 'imapd' as tcpwrapper
 service name
2002.04.15 17:08:38 LOG5[1820:1680]: stunnel 3.22 on
 x86-pc-mingw32-gnu WIN32 with OpenSSL
 0.9.6c 21 dec 2001
2002.04.15 17:08:38 LOG5[1820:1680]: FD_SETSIZE=4096, file ulimit=-1
(unlimited) -> 2000 clients allowed
```

You're already familiar with the `-d` option, but here we've introduced `-l` and `-N`. The `-l` option launches the specified program for each incoming connection. In this case, we launched the imapd daemon. The `-N` is useful, especially on Cygwin systems for forcing a service name for TCPWrappers inspection. The service names are found in the /etc/services file and are necessary to match entries in the /etc/hosts.allow and /etc/hosts.deny files.

APPLICATION INSPECTION

So far we have looked at tools that examine the Web server. In doing so, we miss vulnerabilities that may be present in the Web application. This class of vulnerabilities arises from insecure programming and misconfigurations of the interaction between Web servers and databases. We can't explain the nature of Web application insecurity and the methodology and techniques for finding those vulnerabilities within a single chapter. What we will show are the tools necessary for you to peek into a Web application. Although a few of these programs have grown from the security community, they deserve a place in a Web application programmer's debugging tool kit as well.

Achilles

Aptly named, Achilles helps pick apart Web applications by acting as a proxy with a pause button. A normal proxy sits between a Web browser and a Web server, transparently forwarding requests and responses between the two. Achilles works similarly, but it adds functionality that lets you modify contents on the fly. For example, Achilles lets you manipulate cookie values, POST requests, hidden Form fields, and every other aspect of an HTTP transaction—even over SSL!

Implementation

Because it's a proxy, Achilles must first be set up to listen on a port and placed into "intercept" mode. Figure 8-7 illustrates the basic configuration for starting Achilles in its proxy mode. Clicking the play button (the triangle) starts the proxy, and clicking the stop (square) button stops it—think of a tape recorder's controls.

It's a good idea to leave the Ignore .jpg/.gif option enabled. Modifying image files rarely bypasses a Web application's security stance, and the number of requests it generates from a single Web page quickly becomes annoying.

Next, set your Web browser's proxy to the IP address (127.0.0.1 if it's the same computer) and port (5000, by default) on which Achilles listens. Normally, it's easiest to run Achilles on your localhost. Any Web browser that supports an HTTP proxy, from Lynx to Galeon, can use Achilles. The restriction to the Windows platform is that Achilles is a Win32 binary.

In basic intercept mode, you can browse a Web site or multiple Web sites transparently. The Log To File option will save the session to a file. This is useful for surveying a

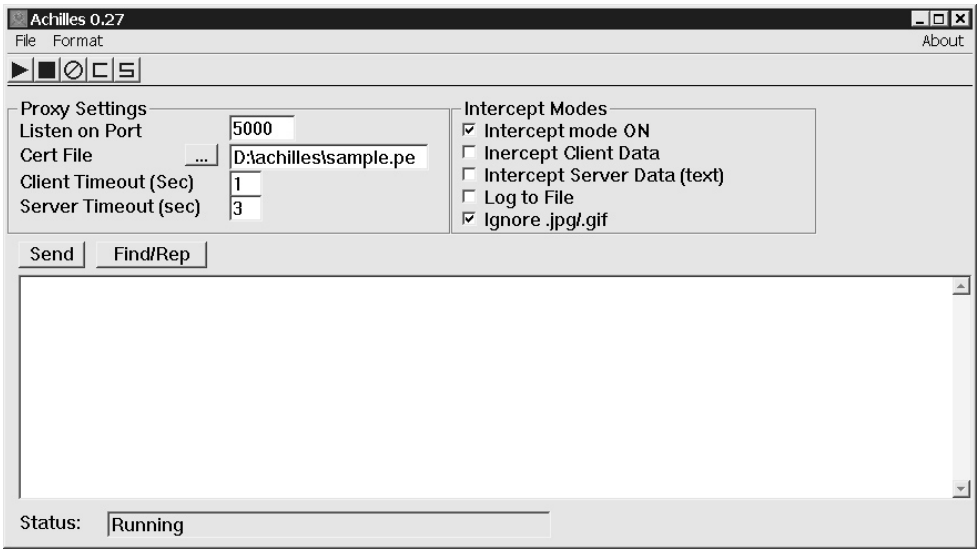

Figure 8-7. Basic proxy settings for Achilles

Web application. The logfile holds every link that was visited, including helper files such as JavaScript (*.js) and other include (*.inc) files that are not normally seen in the URL. The other advantage is that you now have a copy of the HTML source of the target Web site. This source might reveal hidden Form fields, cookie values, session-management variables, and other information about the application. The techniques for picking apart a Web application are well beyond the scope of this chapter, but having a tool like Achilles is an absolute requirement for performing such tests.

In active intercept mode, you can view the requests made by the browser (Intercept Client Data) or responses sent by the server (Intercept Server Data (text)). Intercepting client data enables you to manipulate GET and POST requests as well as cookie values. This capability is used to bypass authentication and authorization schemes, and to impersonate other users. Achilles's text box basically functions as a text editor.

Using Achilles probably sounds abstract by now. This is definitely a tool in the "pictures are worth a thousand words" category. Launch Achilles, change your Web browser's proxy setting, make sure to choose Intercept Client Data, and browse your favorite Web site. You'll be surprised to see what goes on behind the scenes of ordering a book or checking your bank balance!

Interception Problems Achilles intercepts only text data. A site that uses ActiveX components, also known as COM (Component Object Model) objects or CAB (cabinet) files, is more resilient to interception because such files appear as binary data that Achilles always ignores. Achilles still correctly proxies the HTTP connection, but you will not be

able to manipulate the data. Other binary objects, such as downloading a ZIP or PDF file, are also proxied but are not shown in the text window.

Web sites that use SSL often generate errors in Achilles. A problematic site with 20 objects on its home page (such as pictures, style sheets, JavaScript files, and HTML) might generate 20 "Client failed SSL connection" errors. This is not really a big deal, but it does mean that you have to click 20 different OK buttons to close each error indication.

Some sites tend to cause Achilles to crash unexpectedly. There does not seem to be any good rule of thumb that determines which sites cause crashes and which do not. One workaround is to log onto the site with the proxy, and then start the proxy and change your browser's settings once you come to a particular part of the application you wish to inspect. Unfortunately, this technique fails against sites that use strong session management. Finally, Achilles handles HTTP Basic Authentication, but any Web application that uses NTLM Authentication (supported by IIS) will not work through Achilles.

WebSleuth

WebSleuth puts proxy functionality right in the browser. It is a set of Visual Basic routines wrapped around Internet Explorer. Obviously, this ties you to the Win32 platform, but the tool is worth it. It allows you to step through a site while examining cookies and HTML source, taking notes along the way.

Implementation

Figure 8-8 shows the WebSleuth interface, in which several options are available. The raised buttons, Go, Back, Stop, Fwrd, and Edit Source, perform a single function when clicked with the left mouse button. The right mouse button pulls up a menu for each of the flat buttons, Properties, Toolbox, Plugins, and Favorites.

The Toolbox menu button has some of the best functions. The HTML Transformations function is especially cool. It removes scripts, which disables many types of input validation routines. It shows hidden fields, which reveals session, server, and client variables. Also, the Generate Report function creates an excellent list of the current page's cookies, links, query strings, Form information, script references, comments, and META tags.

The Properties menu button displays information about the current page. This is useful for watching when cookie values are set, inspecting query strings, or enumerating the links available on the page.

Final Notes The Analyze… functions under the Options tab do not work over SSL. These functions launch a pop-up window that contains the HTTP request and its arguments, if any. It is at this point that you would modify data to change a POST request, for example. Unfortunately, it does not work!

Wget

The final tool we present probably seems out of place compared to the previous ones. Wget is a command-line tool that basically copies a Web site's contents. It starts at the home page and follows every link until it has discovered every page of the Web site.

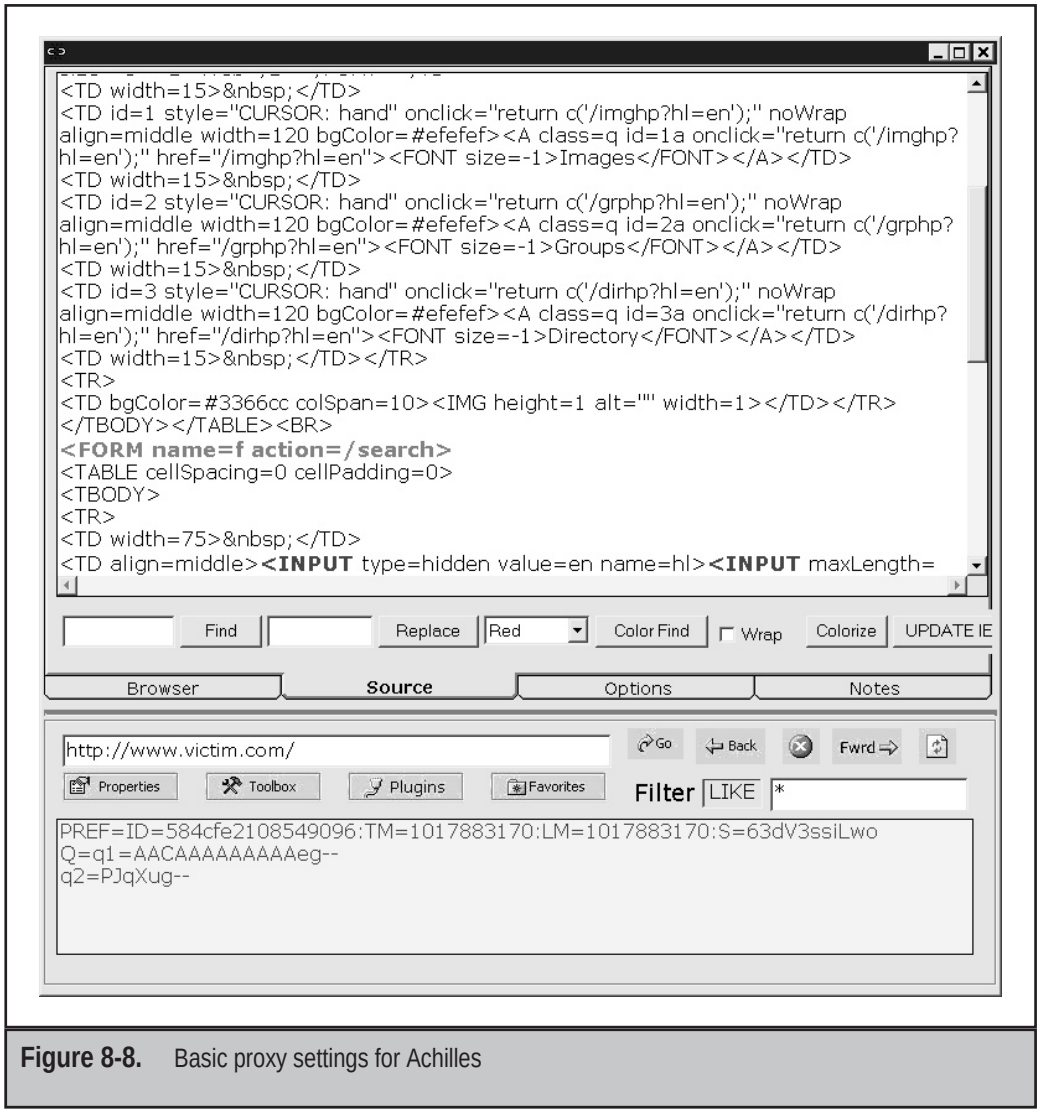

Figure 8-8. Basic proxy settings for Achilles

When someone performs a security audit of a Web application, one of the first steps is to sift through every page of the application. For spammers, the goal would be to find e-mail

addresses. For others, the goal would be to look for programmers' notes that perhaps contain passwords, SQL statements, or other juicy tidbits. In the end, a local copy of the Web application's content enables the person to search large sites quickly for these types of information.

Wget has other uses from an administrator's point of view, such as creating mirrors for highly trafficked Web sites. The administrators for the mirrors of many Web sites (such as *www.samba.org* and *www.kernel.org*) use wget or similar tools to reproduce the master server on alternative servers. They do this to reduce load and to spread Web sites geographically.

Implementation

As wget's main purpose is to download the contents of a Web site, its usage is simple. To spider a Web site recursively, use the `-r` option:

```
$ wget -r www.victim.com
...(continues for entire site)...
```

The `-r` or `--recursive` option instructs wget to follow every link on the home page. This will create a *www.victim.com* directory and populate that directory with every HTML file and directory wget finds for the site. A major advantage of wget is that it follows every link possible. Thus, it will download the output for every argument that the application passes to a page. For example, the viewer.asp file for a site might be downloaded four times:

- viewer.asp@ID=555
- viewer.asp@ID=7
- viewer.asp@ID=42
- viewer.asp@ID=23

The @ symbol represents the ? delimiter in the original URL. The ID is the first argument (parameter) passed to the viewer.asp file. Some sites may require more advanced options such as support for proxies and HTTP Basic Authentication. Sites protected by Basic Authentication can be spidered in this way:

```
[root@meddle]# wget –r --http-user:dwayne --http-pass:woodelf \
> https://www.victim.com/secure/

...continues for entire site...
```

Sites that rely on cookies for session state or authentication can also be spidered by wget. Create a cookie file that contains a set of valid cookies from a user's session. The prerequisite, of course, is that you must be able to log in to the site to collect the cookie values. Then, use the `--load-cookies` option to instruct wget to impersonate that user based on the cookies:

```
$ wget –-load-cookies=cookies.txt \
> –r https://www.victim.com/secure/menu.asp
```

Still other sites purposefully set cookies to defeat most spidering tools. Wget can handle session and saved cookies with the appropriately named `–cookies` option. It is a Boolean value, so you can either turn it off (the default) or on:

```
$ wget –-load-cookies=cookies.txt –cookies=on \
> –r https://www.victim.com/secure/menu.asp
```

The `--http-user` and `--http-passwd` options enable wget to access Web applications that employ HTTP Basic Authentication. Set the values on the command line and watch wget fly:

```
$ wget –-http-user=guest –http-passwd=no1knows \
> –r https://www.victim.com/maillist/index.html
```