The WebGoatv2 User & Install Guide

January 24th, 2003

Webgoat Project is sponsored by

Table of Contents

1. Introduction

Overview

The WebGoatv2 application is designed to illustrate typical security flaws within web-applications. It is intended to teach a structured approach to testing for, and exploiting, such vulnerabilities within the context of an Application Security Assessment.

A full Application Security Assessment testing methodology is being documented by http://www.owasp.org/testing/ and this will provide a superset of the issues demonstrated within the WebGoat. It may include a formal design and code review, for example. The WebGoat tutorials aim to give practical training and examples relating to the *Implementation Review* phase of the OWASP Web Application Security Testing Methodology.

The WebGoatv2 Application provides a testing platform for a typical application security assessment. The assessor is given the same information and rights as a typical customer or client of an on-line application.

- It is a web based application.
- The attack simulations are *remote* – e.g. all of the described techniques may be performed from any connected location.
- Whilst credentials and operational information is provided, the testing is ostensibly *black-box*, as source-code is not supplied[1]

Of course, the teaching aspect of WebGoat means that certain information will be revealed that would not typically be available in order to guide the tester through an assessment process.

The current lesson plans provided for in Webgoatv2 include:

```
➿Http Basics
➿How to Perform Database Cross Site Scripting (XSS)
➿How to Spoof an Authentication Cookie
➿How to Exploit Hidden Fields
➿How to Discover Clues in the HTML
➿How to Perform Parameter Injection
➿How to Perform SQL Injection
➿How to Exploit Thread Safety Problems
➿How to Exploit Unchecked Email
➿How to Spoof an Authentication Cookie
➿Putting it all together
```

Future releases of WebGoat will include more tutorials and functionality. Should you have any suggestions for improvement or new tutorials please contact : bill@owasp.org with your ideas.

---

[1] Although by virtue of the fact WebGoat is Open-Source, this is debateable! The Show Java option has therefore been provided.

2. Objectives

Having followed the testing techniques within WebGoat, a tester should be able to:

- Understand the high-level interaction processes within a web-application,
- Determine information within client visible data which could be useful in an attack,
- Identify and understand data and user interactions which may expose the application to attack,
- Perform tests against those interactions to expose flaws in their operation,
- Execute attacks against the application to demonstrate and exploit vulnerabilities.

3. Tools required

There are a number of tools emerging to aid the wily application security assessor. By far the most relevant to this type of application security assessment are local proxies and web/application spiders.

Application Assessment Proxy

A normal web-proxy typically receives, processes and forwards HTTP and HTTPS traffic between the client and server. This is normally to provide a single point through which all web traffic passes – for example to monitor usage, improve performance through caching or apply security policies.

An application proxy tool is designed to intercept all http and https communication between the local client browser and the server-side. It acts as a *man-in-the-middle* where all interaction may be monitored, reviewed and (importantly) modified.

Through such a tool, the assessor can determine exactly what data is passed between the Client and Server. Furthermore, they may analyse and modify the data in order to test the impact on the application.

It is essential for many of the tests within WebGoat that an application assessment proxy, or software with equivalent functionality, be used.

The following is recommended:

- OpenProxy – http://www.owasp.org

Application Spider

Spidering or crawling a site should identify and follow all of the intended pages and links within a web site & application, and optionally store a local copy.

The results can then be analysed to define a comprehensive list of target scripts, forms, pages and fields within the application for use in later testing.

Mirrored content can also be analysed for relevant information far more quickly than through a manual or 'on the wire' analysis.

The following are recommended:

- HTTrack – www.httrack.com
- Form Scalpel – http://www.ugc-labs.co.uk/tools/formscalpel/

4. Using WebGoat

Having identified the objectives and required tools, we may now get started with the WebGoat lessons.

5. Installation

WebGoat is a platform independent environment.

It utilises Apache Tomcat and the JAVA development environment.

Installers are provided for Microsoft Windows and UN*X environments, together with notes for installation on other platforms.

```
INSTALLATION:

1. install the Java Development Kit (JDK) from
   http://java.sun.com/j2se/1.4/
2. set JAVA_HOME=c:\j2sdk1.4  (or wherever you installed it)
3. install the Tomcat application server from http://jakarta.apache.org/
4. drop the WebGoat-*.war file into the "webapps" directory inside the
   tomcat directory
5. rename WebGoat-*.war to WebGoat.war in the "webapps" directory
6. edit tomcat/web.xml and uncomment the invoker servlet mapping

INSTALLATION using unix/windows installer:

1) install_WebGoat-x.x-dev_unix.jar

        Must have a Java runtime environment installed
        java -jar install_WebGoat-x.x-dev_unix.jar

2) install_WebGoat-x.x-dev_windows.jar

        Must have a Java runtime environment installed
        java -jar install_WebGoat-x.x-dev_unix.jar


BUILDING: (skip if you just want to run)

1. install ant from http://jakarta.apache.org
2. put tomcat/server/lib/catalina-ant.jar into your ant/lib directory
3. unpack the WebGoat src distribution
4. modify catalina.home property in build.xml to specify tomcat
   installation directory
5. add '<user username="webgoat" password="webg0@t"
   roles="admin,manager,standard,tomcat"/>'
   to the tomcat_home/conf/tomcat-users.xml file
6. type 'ant' to start the compile
7. type 'ant dist' to create a new WebGoat .war file in the dist
   directory
8. type 'ant install' to install the current build directory in tomcat.
   This does not install the application permanently -- you have to put
   the .war file in the webapps directory  for that.
9. type 'ant reload' to reinstall the current build directory in tomcat
```

RUNNING:

1. go to the tomcat/bin directory
2. click on startup.bat (set properties on the terminal window to 5000 lines)
3. start your browser and browse to...

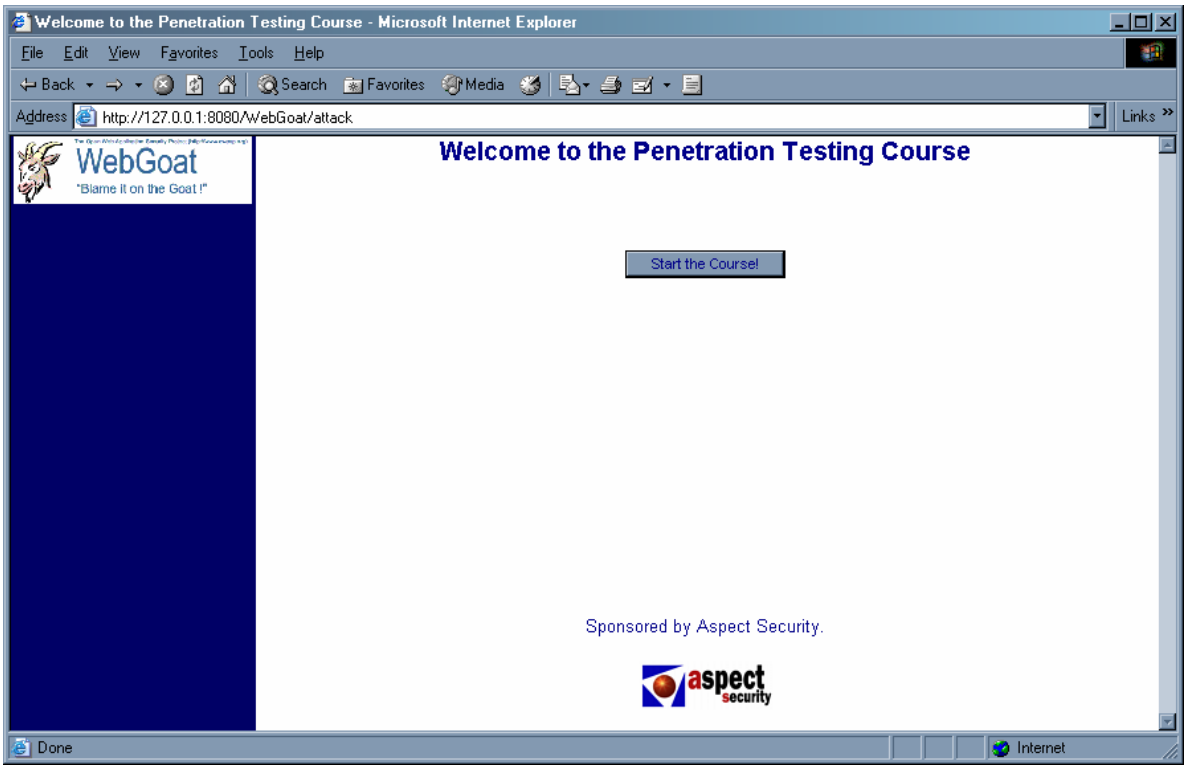
http://localhost:8080/WebGoat/attack

6. Getting Started

In order the start using WebGoat, Tomcat must be launched using the `startup` script/bat in the Tomcat `bin` directory.

From a browser, the Tomcat server can be accessed on localhost port 8080, e.g. `http://127.0.0.1:8080`

WebGoat resides in the WebGoat directory, and the lessons can be found at: http://127.0.0.1:8080/WebGoat/attack

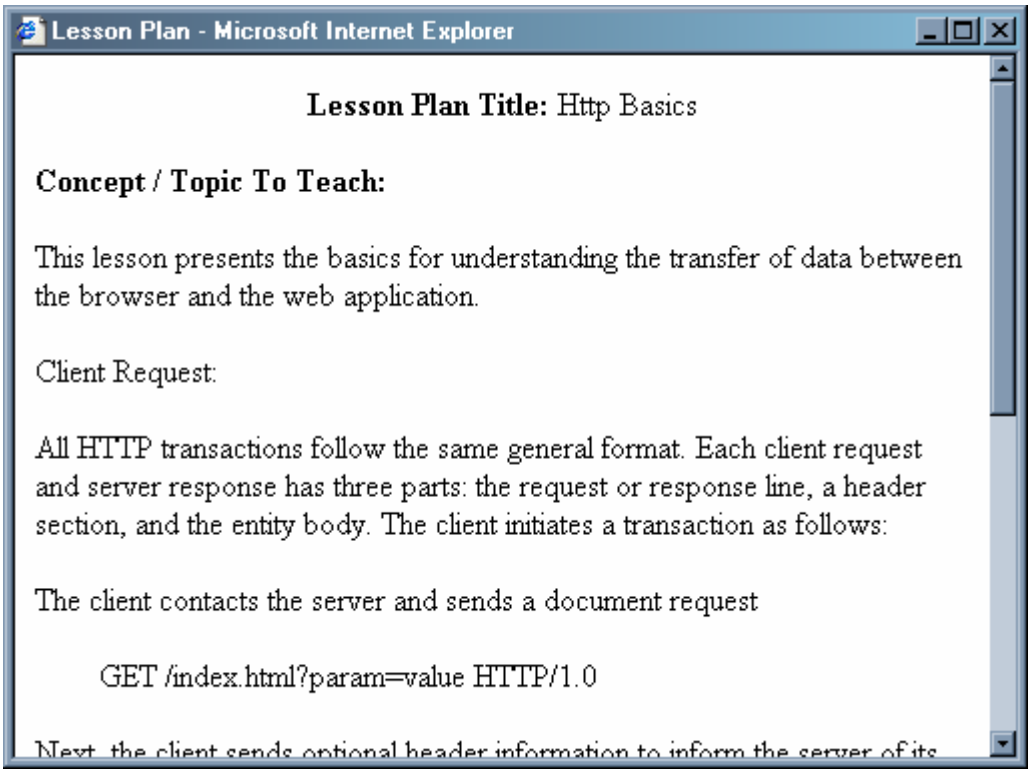


7. Lesson Plans

The current Lesson Plans included within this relase of WebGoatv2 include

- Http Basics
- How to Perform Database Cross Site Scripting (XSS)
- How to Spoof an Authentication Cookie
- How to Exploit Hidden Fields
- How to Discover Clues in the HTML
- How to Perform Parameter Injection
- How to Perform SQL Injection
- How to Exploit Thread Safety Problems
- How to Exploit Unchecked Email
- How to Spoof an Authentication Cookie
- Putting it all together

For each lesson within WebGoat, an overview and objectives are provided, and are accessed through the *Show Lesson Plan* button.



These lesson plans describe the operation of each aspect of the target application, the areas of interest relating to the security assessment and provide some indication of the type of attack that should be attempted.

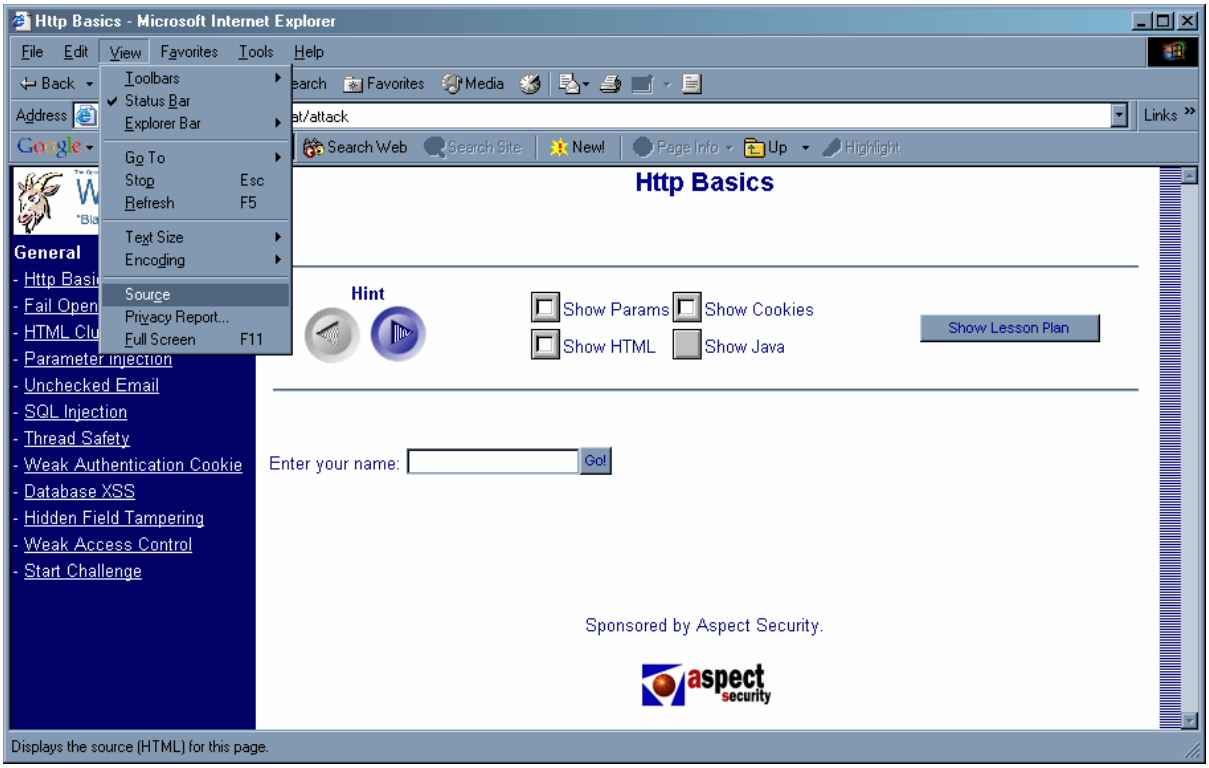
8. Basic operation

At each stage of an application security assessment, it is important to understand the operation of the target.

This typically involves:

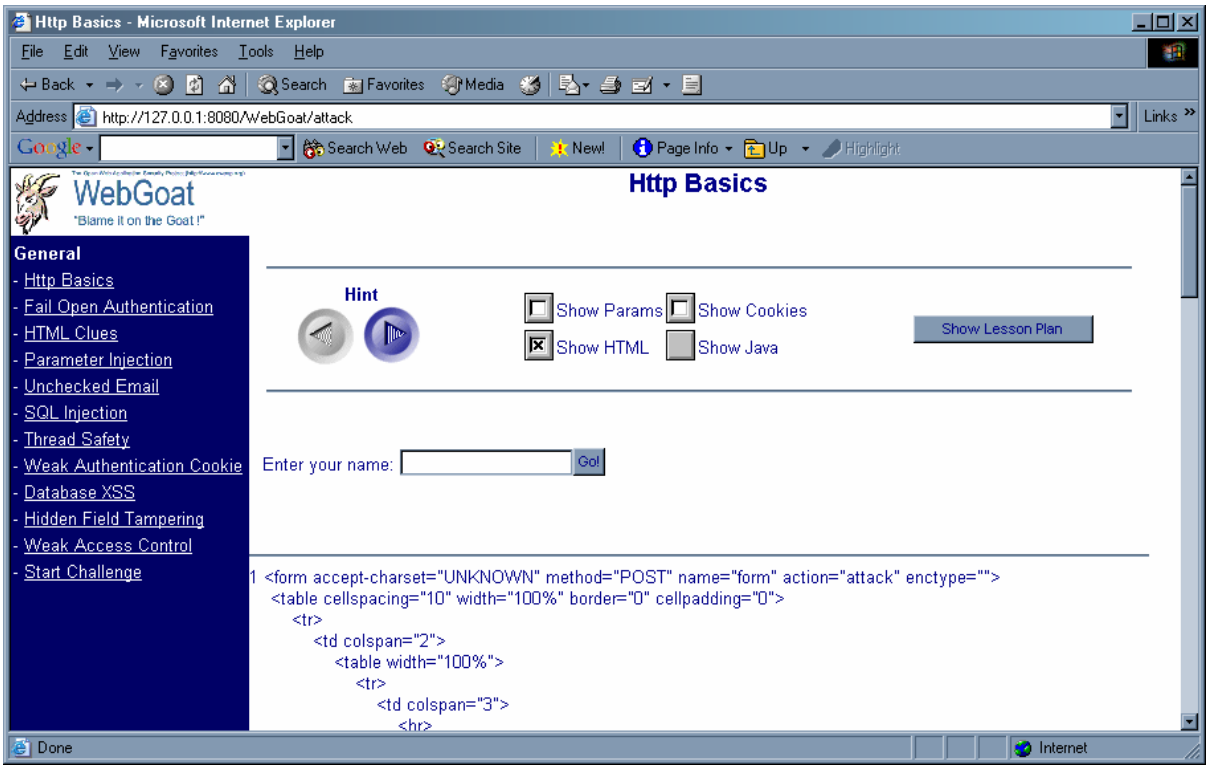
- Examining client side content, such as HTML and script,
- Analysing communications between the client and server,
- Reviewing cookies and other local data.

Under normal circumstances, HTML and other client-side data could be viewed using the browsers own features:

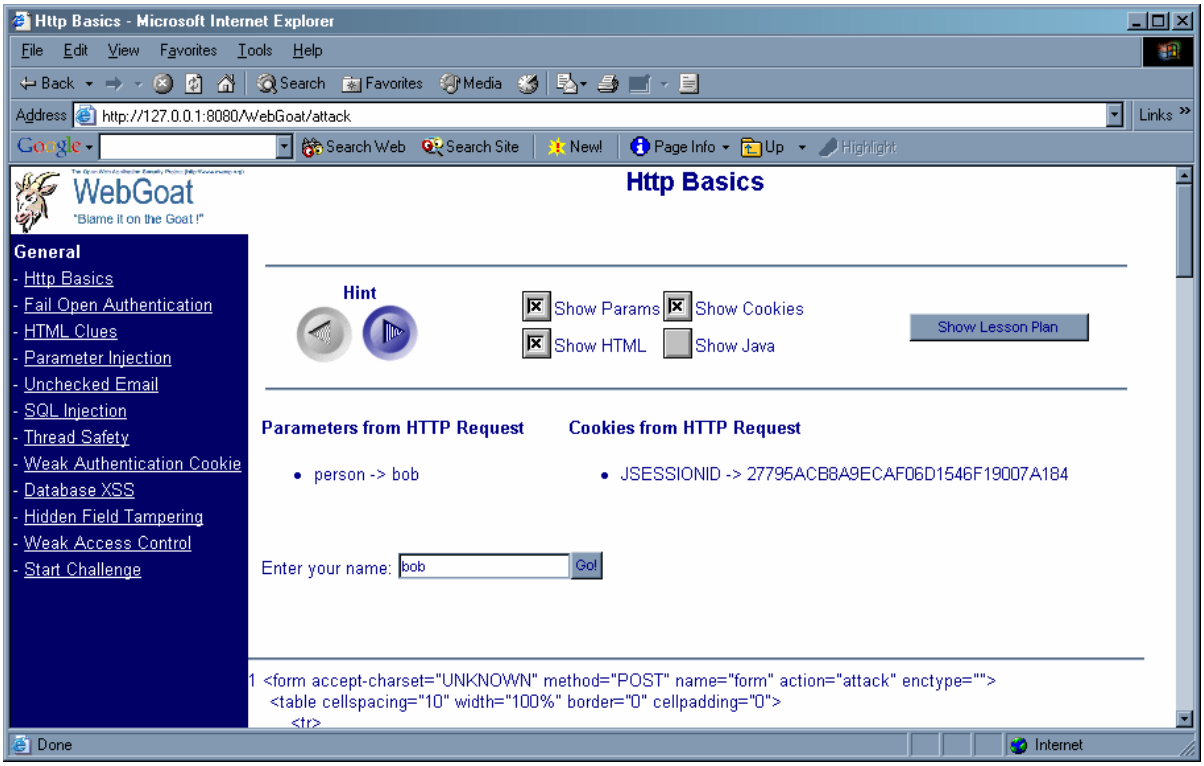


WebGoat provides a wrapper around the test application. This source would therefore also include the side and top bars, and other information which is not relevant to the lessons.

For this reason, the Show HTML tag has been provided. This shows only the content relating to the target application.



Similarly, form parameters and cookies can be displayed:



Furthermore, the luxury of being able to examine the server-side source has been provided in WebGoat, through the Show Java option.

9. Proxy Utilisation

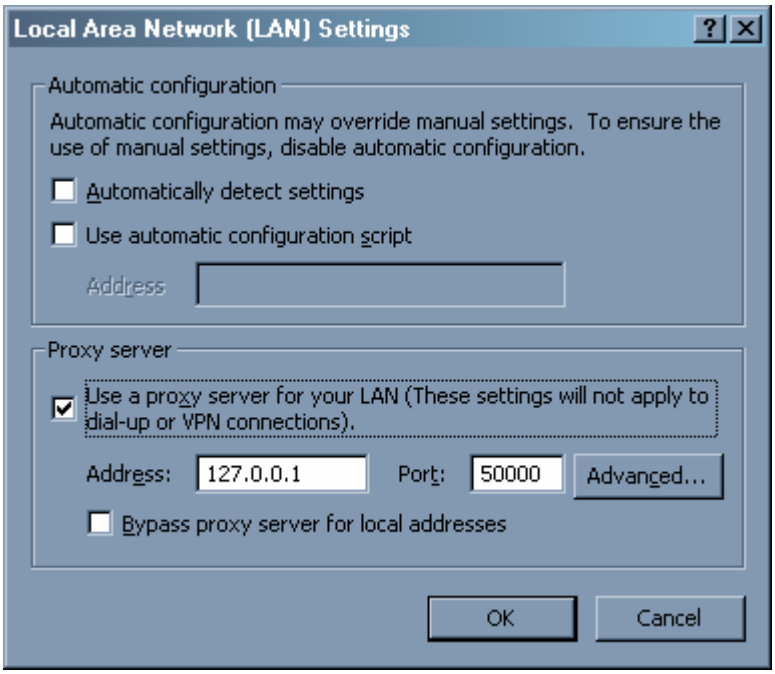
The features within WebGoat for revealing the workings of the application also need to be supplemented with the assessor's application assessment proxy of choice.

This enables further analysis and modification of the client-server interaction, and data in transit.

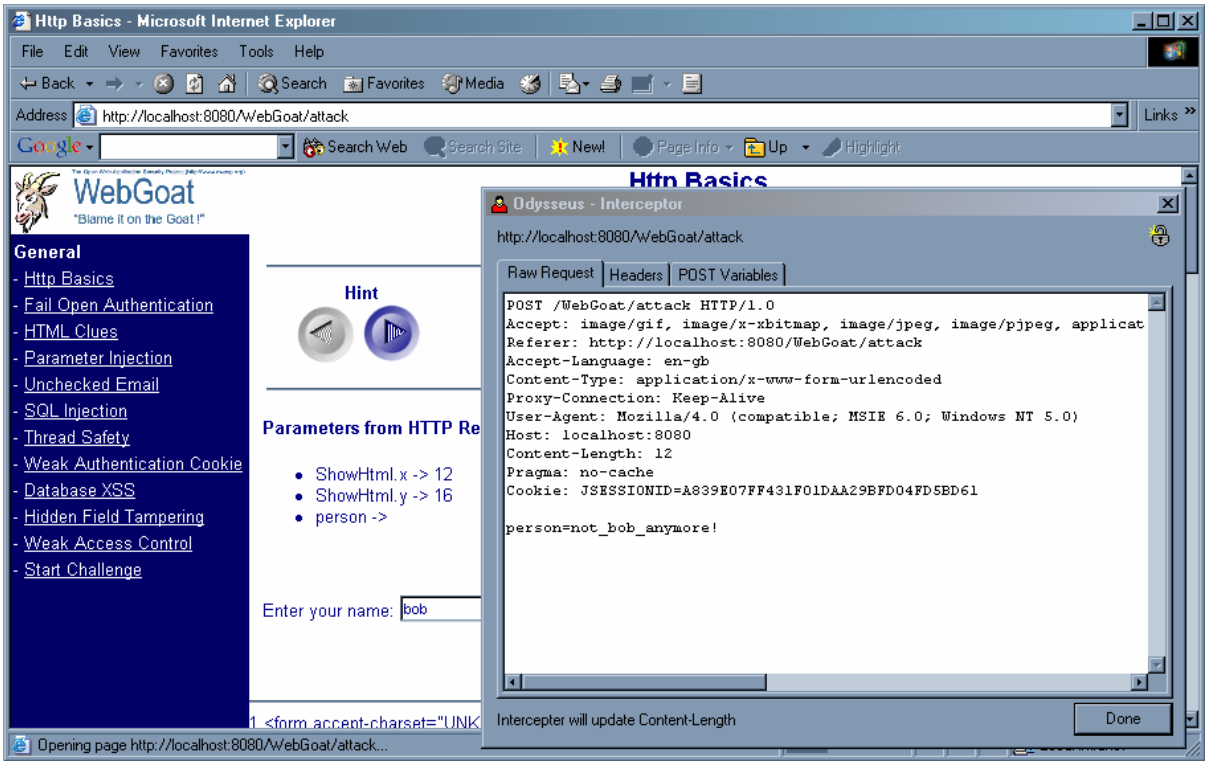
Use and configuration will be unique to each tool, but the basic concepts are as follows:

- The application assessment proxy must sit *between* the client browser and the remote server
- It should allow the display and modification of all HTTP data in transit

Typically, the tool will either plug directly into the browser, or listen on another local port. In the former case, a special URL may need to be entered in the browser; in the latter the browser will probably need to be configured to use the proxy as it would any other, e.g.



Now, when receiving or sending data from the client browser, we can intercept, analyse and modify it to test for security flaws in the target.



In using proxies of this kind, a number of capabilities become available to the assessor, including:

- All GET/POST parameters are available for modification, regardless of their hidden status,
- Cookies, both persistent and non-persistent may be modified when entering or leaving the browser,
- All client-side data validation can be bypassed, as the parameters can be modified immediately before being sent to the server,
- Information regarding the caching of data (e.g. Pragma: no-cache) is exposed for analysis,
- Server: and other headers are revealed, which may be useful for enumeration of the remote web- and application- server technologies.

10. Ready to Go!

All that remains is to start the WebGoat lessons.

One last point – if the problem or solution don't reveal themselves to you, there are hints available to guide you through the lessons. Don't be too eager, though – application testing is 10% technique and 90% lateral thinking. You can blame it on the Goat, but you can't rely on him!

11. How to write a new WebGoat lesson

All you have to do is implement the abstract methods in LessonAdapter.

WebGoat uses the Element Construction Set from the Jakarta project. You should read up on the API for ECS at

http://www.peerfear.org/alexandria/content/html/javadoc/ecs/HEAD/index.html.

In addition you can look at the other lessons for examples of how to use the ECS.

Step 1: Set up the framework

```
import java.util.*;
import org.apache.ecs.*;
import org.apache.ecs.html.*;

/**
 *  Copyright (c) 2002 Free Software Foundation developed under the custody of the Open Web
 *  Application Security Project (http://www.owasp.org) This software package is published by
 *  OWASP under the GPL. You should read and accept the LICENSE before you use, modify
 *  and/or redistribute this software.
 *
 *   @author     jwilliams@aspectsecurity.com
 *   @created    November 6, 2002
 */
 public class NewLesson extends LessonAdapter
        {

            protected Element createContent(WebSession s)
            {
                return( new StringElement( "Hello World" ) );
            }

            public String getCategory()
            {
            }

            protected List getHints()
            {
            }

            protected String getInstructions()
            {
            }

            protected Element getMenuItem()
            {
            }

            protected Integer getRanking()
            {
            }

            public String getTitle()
            {
            }
        }
```

Step 2: Implement createContent

Creating the content for a lesson is fairly simple. There are two main parts:

(1) handling the input from the user's last request,

(2)generating the next screen for the user.

This all happens within the createContent method. Remember that each lesson should be

handled on a single page, so you'll need to design your lesson to work that way.

A good generic pattern for the createContent method is shown below:

```
// define a constant for the field name
private static final String INPUT = "input";

protected Element createContent(WebSession s)
{
	ElementContainer ec = new ElementContainer();
	try
	{
		// get some input from the user -- see ParameterParser for details
		String userInput = s.getParser().getStringParameter(INPUT, "");

		// do something with the input
		//   -- SQL query?
		//   -- Runtime.exec?
		//   -- Some other dangerous thing

		// generate some output -- a string and an input field
		ec.addElement(new StringElement("Enter a string: "));
		ec.addElement( new Input(Input.TEXT, INPUT, userInput) );
	}
	catch (Exception e)
	{
		s.setMessage("Error generating " + this.getClass().getName());
		e.printStackTrace();
	}
	return (ec);
}
```

ECS is quite powerful -- see the Encoding lesson for an example of how to use it to create a table with rows and rows of output.

Step 3: Implement the other methods

The other methods in the LessonAdapter class help the lesson plug into the overall WebGoat framework. They are simple and should only take a few minutes to implement.

```
  public String getCategory()
{
        // The default category is "General" Only override this
        // method if you wish to create a new category or if you
        // wish this lesson to reside within a category other the
        // "General"

        return( "NewCategory" );  // or use an existing category
}

protected List getHints()
{
        // Hints will be returned to the user in the order they
        // appear below.  The user must click on the "next hint"
        // button before the hint will be displayed.

        List hints = new ArrayList();
        hints.add("A general hint to put users on the right track");
        hints.add("A hint that gives away a little piece of the problem");
        hints.add("A hint that basically gives the answer");
        return hints;
}

protected String getInstructions()
{
        // Instructions will rendered as html and will appear below
        // the area and above the actual lesson area.
        // Instructions should provide the user with the general setup
        // and goal of the lesson.

        return("The text that goes at the top of the page");
}

protected Element getMenuItem()
{
        // This is the text of the link that will appear on
        // the left hand menus under the appropriate category.
        // Their is a limited amount of horizontal space in
        // this area before wrapping will occur.

        return( "MyLesson" );
}

protected Integer getRanking()
{
        // The ranking denotes the order in which the menu item
        // will appear in menu list for each category.  The lowest
        // number will appear as the first lesson.
```

```
        return new Integer(10);
}

public String getTitle()
{
        // The title of the lesson.  This will appear above the
        // control area at the top of the page.  This field will
        // be rendered as html.

        return ("My Lesson's Short Title");
}
```

Step 4: Build and test

Once you've implemented your new lesson, you can use ant to build and deploy your new web application. First you want to remove the webgoat .war *AND* the webgoat directory from your webapps directory. Then, from your webgoat directory, type:

```
> ant install
```

This will compile your new lesson and "install" the path into Tomcat. You only need to "install" once. If you make changes to the web application and want to test them, you can use:

```
> ant reload
```

Step 5: Give back to the community

If you've come up with a lesson that you think helps to teach people about web application security, please contribute it by sending it to the people who maintain the WebGoat application.

Thanks!