informIT

An Introduction to Security Testing with Open Source Tools

Date: Sep 16, 2005 By Michael Kelly.

Michael Kelly reports on handy security uses for four open source tools: WebGoat, Firefox Web Developer, WebScarab, and Ethereal. By combining the tools in easy ways, testers can track down and close the gaping security holes that are often left in applications.

The Accidental Tester

I remember my first security bug. It was so simple, I stumbled over it accidentally. (Well, I told the very angry people who were upset with me that it was an accident.) The problem started with a developer who had left his or her user ID in a code comment on the login page for a production system. It looked something like Listing 1.

Listing 1 Source code for the login screen.

```
<TABLE cellSpacing = 0 cellPadding = 1 border = 0 width = "98%" align = "center">
 <TBODY>
 <tr>
  <td class = "wb">
  <table border = 0>
   <FORM method = "POST" name = "login" action = "/login"
   enctype = "application/x-www-form-urlencoded">
    <tr>
    <td class = sb>
     login
     <input type = "text" name = "login_id" size = "32" maxlength = "64"/>
    </td>
    <td class = small>
     enter your username
    </td>
    </tr>
    <tr>
    <td class = sb>
     password
     <input type = "password" name = "password" size = "16" maxlength = "32"/>
    </td>
    <td class = small>
     enter your password
    </td>
    </tr>
    <tr>
    <td class = sb align = right>
     <input type = "submit" name = "submit" value = "login"/>
     <!-- u2x34t - Oct 12, 2004: Removed link to defect tracking system-->
    </td>
    </tr>
   </form>
  </table>
  </td>
 </tr>
 </TBODY>
</TABLE>
```

In case you don't read HTML, note that the comment is enclosed in the <!-- --> tags. Out of curiosity, I entered the user ID (u2x34t) from the source code into the username field and tried to guess the password. I was rewarded with this:

```
The password you entered is incorrect. Please try again.
```

I say *rewarded* because the first user ID I tried gave me this error:

```
The user id you entered is not recognized. Please try again.
```

The specificity of the error messages for the system indicated that I was on the right track. I wouldn't have known that if the system had consistently displayed an error message similar to this one:

```
The user id and/or password you entered are incorrect, please try again.
```

At this point, I knew I had a valid user ID, but I still didn't have the password. I didn't want to simply guess because I didn't want to lock the ID (earlier tests had shown that to be a problem). Instead, I started to wonder about that comment the developer made in the source code:

```
Removed link to defect tracking system
```

I asked myself some questions: What tracking system? How did they remove it? And where did it go? I looked at the source for more clues, but none could be found, at least to my untrained eye. I needed more source code. I figured that if the developer left comments on the login page, there was a good chance he or she left them in other code as well. At the bottom of the login page was a link to a help page. ("Need help logging in? Forgot your ID?") I followed that link and looked at the source for the help screen, where I found something similar to Listing 2.

Listing 2 Source code for the help page.

```
<TABLE cellSpacing = 0 cellPadding = 1 border = 0 width = "98%" align = "center">
 <TBODY>
 <tr>
  <td class = "wb">
  <table border = 0>
   <tr>
   <td class = sb align = right>
    <!--<a href="http://URLForDefectTrackingSystem/%userIdParameter%.aspx">
    Submit a ticket.</a>-->
   </td>
   </tr>
   <tr>
   <!--Help text was here...-->
   </tr>
  </table>
  </td>
 </tr>
 </TBODY>
</TABLE>
```

To remove the link to the defect tracking system, the developer had simply commented out the link. Not only that—the link included a parameter for the user currently logged in so it would know who submitted the ticket. At that point, I had a URL that required a user ID, and I had a user ID. I simply copied the URL, pasted it into the address bar, typed the user ID in the appropriate place, and hit Enter. The system displayed an error page for a defect tracking system, stating that the system was no longer in use. I initially thought I had hit a dead end, but then I saw a link at the bottom of the page: "Return to application." I clicked the link and was rewarded with the home page for the application I was attempting to access. No password required!

After that, I was hooked, and I had to learn more about security testing. That day I bought a copy of How to Break Software Security (Addison-Wesley, 2003, ISBN 0321194330) by James A. Whittaker and Herbert H. Thompson. I'd like to say that I'm a famous white-hat hacker now, but I'm not. I'm just a tester who knows a little bit about security testing. I would still recommend leaving the high-risk testing to the pros, but I know how to find really obvious security bugs, and I'm always looking to learn more. If you're similarly intrigued, this article is for you. We'll review some helpful open source tools that you can download to help get you started with security testing.

Don't worry, you don't need to know anything about security testing to use these tools—in fact, one even counts on it. All you need is some patience and the desire to learn.

WebGoat and Firefox Web Developer

The first trick to learning security testing is selecting an application to help you find security bugs—without getting arrested. One such application, WebGoat, is a full J2EE web app developed and maintained by the Open Web Application Security Project (OWASP). OWASP designed WebGoat to teach web-application security lessons. WebGoat provides a safe and legal environment in which you can practice your testing. It's divided into lessons that teach users how to exploit a real vulnerability on the local system. The value of WebGoat is that it provides not only a safe environment, but hints to help you with your testing. It offers four levels of hints and has features that show the user cookies, parameters, and the underlying HTML and Java code if desired.

The second tool we'll consider is the Web Developer extension for Mozilla Firefox. For a number of reasons, Web Developer is a must-have for any web application tester, but in

this article we'll just look at some of the features that help test security. I regularly use these features, for example:

- Disable cookies
- Convert GET to POST and vice versa
- Make form fields writable
- Display form details
- View cookie information
- View response headers
- Clear cache/authentication/cookies
- Show comments

We'll also examine two other open source tools a little later, but for right now let's focus on some WebGoat and Web Developer examples. To use WebGoat, you might need to install Tomcat and the latest Java Development Kit (JDK). It can take some time to get WebGoat set up, but I encourage you to do it now if you want to follow along. The following examples use WebGoat and Web Developer together.

HTML Clues

Let's start with the HTML Clues example in WebGoat. As illustrated by my security testing experience described earlier, developers can sometimes leave meaningful information in the source code when they don't think anyone will be looking at it, or when a company coding standard forces them to leave comments in the source code when updating. By reviewing the source code for comments such as usernames, passwords, backdoors, or even commented-out links, we can learn a lot about what the developer was thinking when writing the code (or maintaining it).

Figure 1 shows the sign-in form from the WebGoat HTML Clues example.

Figure 1 Sign-in form from the WebGoat HTML Clues example.

Let's say you wanted to look at the source code to read the comments. You could right-click and select View Source, but then you have to know what a comment looks like. What if you don't know how to read HTML? Or what if the HTML is so complex that it's really hard to find and read all the comments? In the Web Developer toolbar is a Miscellaneous menu. If you select it, you can then select Show Comments (see Figure 2).

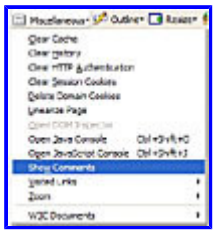

Figure 2 Showing comments in Web Developer.

This command will insert an exclamation point (!) icon on the screen wherever a comment appears in the code. If you click the exclamation point icon, you'll see the comment that the developer left in the code (see Figure 3).

Figure 3 Comments in the WebGoat HTML Clues example.

Just as in my experience described earlier, the developer has left a password in a comment. The purpose of this exercise is to get you to recognize the fact that information like this is sometimes included in the source code. WebGoat is also kind enough to give you the password, because the point of the exercise is simply to get you looking at the code for comments. If you enter the username and password, you get your reward, as shown in Figure 4.

Figure 4 Successful completion of the HTML Clues example.

Ah, the sweet smell of success. It's scary to know that sometimes it really is just that easy.

Hidden Field Tampering

Developers sometimes use hidden fields to maintain information about your session that *they* need to remember, but don't want *you* to see. For example, sometimes a loaded page will include hidden credentials, pricing information, or details about items ordered. This can be an easy way for developers to track information, but it's similarly easy for a hacker to edit the info. A couple of years ago at a conference, Herbert Thompson (coauthor of *How to Break Software Security*) related an experience in which he did something similar to this little trick, to the tune of a couple thousand dollars on a popular web site. It was only through his kind nature that the owners of the site were made aware of the mistake and were able to fix the problem. It was a compelling story.

The form in Figure 5 is from the Hidden Field Tampering example in WebGoat.

Figure 5 Shopping cart from the Hidden Field Tampering example in WebGoat.

Now let's take a look behind the scenes. Again, you could view the source code to see what's hiding in there, but that assumes you can read the code. Worse, it doesn't allow you to change the values even if you do find a hidden field. Traditionally, you would have to use a program to intercept and change the hidden field value that you found in the source code. Web Developer has made all this easy. On the menu, select Forms, Display Form Details (see Figure 6).

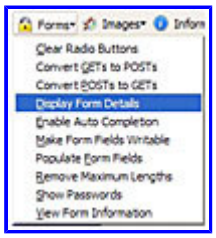

Figure 6 Displaying form details in Web Developer.

This option will show you all sorts of details about the page currently displayed. One bit of information displayed (and editable) is the hidden fields on the page, as shown in Figure 7.

Figure 7 Shopping cart with form details displayed.

If you change the value of the Price hidden field, you'll change the value sent to the server when you click Purchase. For example, if I enter the value 2.00 and submit the page, I get the result in Figure 8.

Figure 8 Successful completion of the Hidden Field Tampering example.

For only $2.00, I might even purchase a second HDTV for my basement!

We could look at some other examples, but I think you get the idea. WebGoat is a learning tool, and Web Developer is a handy tool to have around because it simplifies many of the things that we need to do at the source code level.

Now let's switch gears and examine another type of tool.

WebScarab and Ethereal

WebScarab (also by OWASP) is a framework written in Java for analyzing applications that

communicate using the HTTP and HTTPS protocols. WebScarab records the requests and responses that it observes, and allows you to review them in various ways. The real work is done using security testing plug-ins. At the time of this article, WebScarab had the following plug-ins available (descriptions are largely from the WebScarab site):

- **Fragments.** Extracts scripts and HTML comments from HTML pages as they are seen via the proxy or other plug-ins.
- **Proxy.** Observes traffic between the browser and the web server. The WebScarab proxy is able to observe both HTTP and encrypted HTTPS traffic by negotiating an SSL connection between WebScarab and the browser, instead of simply connecting the browser to the server and allowing an encrypted stream to pass through it. Various proxy plug-ins have also been developed to allow the operator to control the requests and responses that pass through the proxy.
- **Spider.** Identifies new URLs on the target site and fetches them on command.
- **Manual Request.** Allows editing and replay of previous requests, or creation of entirely new requests.
- **SessionID Analysis.** Collects and analyzes a number of cookies (and eventually URL-based parameters) to visually determine the degree of randomness and unpredictability.
- **Scripted.** Operators can use BeanShell (a lightweight scripting language for Java) to write a script to create requests and fetch them from the server. The script can then perform some analysis on the responses, with all the power of the WebScarab Request and Response object model to simplify things.
- **Parameter Fuzzer.** Performs automated substitution of parameter values that are likely to expose incomplete parameter validation, leading to vulnerabilities such as cross-site scripting (XSS) and SQL injection.

Ethereal is a network packet analyzer that can read more than 706 protocols directly "off the wire" from a live network connection. Ethereal is an excellent tool for examining security problems, debugging protocol implementations, and for learning the internals of a given protocol. Ethereal and WebScarab overlap in functionality, but I mention Ethereal because it works with much more than just HTTP and HTTPS.

There is more to WebScarab and Ethereal than I can show in one article, but I hope to entice you enough to download and start to learn more about them on your own.

Fail-Open Authentication

This example will show how to spoof an authentication cookie in WebGoat using WebScarab. According to OWASP, the security term *fail-open* describes the situation when an error occurs during a verification method, causing that method to evaluate to true. The authentication is passive in that, if the password is not provided, the system won't bother to check for it or even ask why it wasn't provided.

In their paper "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection," Thomas H. Ptacek and Timothy N. Newsham talk about fail-open in terms of firewalls:

The terms "fail-open" and "fail-closed" are most often heard within the context of firewalls, which are access-control devices for networks. A fail-open firewall stops controlling access to the network when it crashes, but leaves the network available. An attacker that can crash a fail-open firewall can bypass it entirely. Good firewalls are designed to "fail-closed", leaving the network completely inaccessible (and thus protected) if they crash.

Our example will look at this phenomenon during an application login. Figure 9 shows the login screen in WebGoat.

Figure 9 Login from the Fail Open Authentication example in WebGoat.

By running WebScarab and configuring my browser to point to a local proxy, I can log in without a password. If I simply enter the user name webgoat and click Login, WebScarab gives me the result shown in Figure 10.

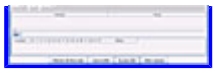

Figure 10 Intercepted HTTP POST in WebScarab.

If I delete the text &Password= from the last line in the POST and click Accept Edits, I get the result in Figure 11. (Note that I click Release All Intercepts on the response from the server.)

Figure 11 Successful completion of the Fail Open Authentication example.

Installing WebScarab, configuring my browser to perform this test, and running this test took me a total of 15 minutes. Tools like WebScarab and Ethereal are intimidating to people who have never used them before, but they're *powerful*. Don't be intimidated. Take the 15 minutes to run this test on your own. Then play around with your current application and see what you can find out using WebScarab.

Secure Sockets Layer (SSL)

In this example, we'll use Ethereal to check the SSL on my email server. I ran this test a couple of months ago with Jonathan Bach when someone told us that they could hack our email by using Ethereal. Jonathan and I ran a series of tests:

- Using his computer, could Jonathan capture the network traffic from my wireless card?
- Could Jonathan see my password when I logged into my email without using SSL?
- Could Jonathan see my password when I logged in using SSL?

Figure 12 shows the login screen for my email. Notice that SSL is optional.

Figure 12 Login screen for SSL example.

In this example, I have Ethereal running on a different computer (just as Jonathan did). This is very different from using WebScarab, because Ethereal is not using a proxy server. Ethereal is just capturing the information that my wireless network card sends out for the entire world to see. If I attempt to log in without using SSL, Figure 13 shows what I record on the second computer.

Figure 13 Login information in Ethereal.

If you look closely at Figure 13, you'll see the username (userName) and password (myPassword) that I used for this example. Figure 14 shows a closeup.

Figure 14 Username and password.

If I run the same test using SSL, I can't find my username and password anywhere. I find a lot of entries that look similar to the ones in Figure 15, and if I perform a search for the password, it comes back as "match not found."

Figure 15 Login SSL information in Ethereal.

Good enough for me. Someone may be able to break the SSL encryption, but my email isn't really all that interesting, so I'm not too worried about it. Either way, that's outside of the introductory scope of this article.

As a public service announcement, be aware that if you ever check your email at a public wireless access point—for example, at a conference or the hotspots at your local coffeehouse—you're probably exposing your data to nefarious people like me. (Don't believe me? Check out what happened at the 2005 Defcon conference.) Either avoid doing anything you don't want other people to observe, or use a tool like Anonymizer to protect yourself. (Thanks to James Bach for the Anonymizer tip.)

Next Steps

I regularly use the four open source security testing tools covered in this article. (I mostly use WebGoat for examples.)

For the absolute beginners out there, a great resource is Hacker Highschool. As the name implies, this material was designed for high school students. If you think you need to start at square one, start here. If you find the material useful, consider contributing to the worthy cause.

For some less basic but still introductory material, check out Julian Harty's work on Commercetest.com. Julian provides open content for nonfunctional testing (including security testing). As mentioned earlier, I like the Whittaker and Thompson book How to Break Software Security; I also recommend the presentation "Top Web App Attack Methods and How to Combat Them," by Dennis Hurst of SPI Dynamics.

Finally, once you feel ready to jump in, try some work with OWASP. They have tools, advanced material, and plenty of opportunities for you to get involved. In addition, check out Insecure.org, a great source for news, tools, and instructions if you're serious about security testing.

NOTE

This article was written using the notes from a meeting of the Indianapolis Workshop on Software Testing, held in June 2005 on the topic of "Open Source Testing Tools." Participants in the workshop included Taher Attari, Charlie Audritsh, Mike Goempel, Michael Kelly, Marc Labranche, Jeffery Mexin, Patrick Milligan, Richard Moffatt, Dana Spears, and Jon Strayer.
