Using Libwhisker

by Neil Desai
last updated August 24, 2004

As noted in the article "Penetration Testing of Web Applications" the use of web applications to conduct business is increasing. Companies often have custom sites built by in-house developers, and it is almost impossible to find all the vulnerabilities in a web site using automated tools. Simply looking for default installations of different software may turn up nothing, but it may still be vulnerable to many different programming errors in this custom-built site. Conducting an assessment of website can be a major undertaking and it is much more painful if the assessment is carried out with out the proper tools. A manual inspection of the site is almost always required, but when a particular vulnerability is found it can be very handy to have a set of tools to automate certain steps from there.

Why Libwhisker?

Since we are dealing with custom applications we need a set of custom utilities. There are many different tools out there that can be scripted, but we are going to focus on Libwhisker. Libwhisker is not a tool or application in itself; it is a PERL library which allows for the creation of custom HTTP packets.

Since Libwhisker is a PERL module and not an application, it is assumed that the reader has some knowledge of the HTTP protocol and is familiar with writing PERL scripts that use external modules.

First let's answer the question: why do we do we need to look at another PERL module to do what can all ready be done through existing PERL modules (i.e. LWP, HTTP, URI)? Libwhisker offers us many advantages over other PERL modules:

1. It is 100% PERL code. This means that it is completely portable to any system that runs PERL.
2. It combines the functionality of many different PERL modules normally used for interacting with web applications.
3. It does not need to be installed. Just have a copy of the "LW2.pm" in the same directory as the script that needs to use it.
4. It was created with a no rules approach. This means that it allows the user to create requests that don't conform to the standards. Other PERL modules may try and enforce standards on our requests.

Using Libwhisker

The main data structure in Libwhisker is the 'whisker' anonymous hash. A hash is a data structure in PERL that is comparable to associated arrays in other programming and scripting languages.

This 'whisker' anonymous hash can either define different aspects of a HTTP request or read different parts of the HTTP response. However, determining how to access this information can be a source of confusion.

Prior to using any of the Libwhisker functions, two PERL hashes first need to be defined, one for the HTTP request and one for the HTTP response. Some items will be defined in the 'whisker' hash and some will be either defined directly in the request hash or read directly from the response hash. To determine which portions of the HTTP request/response are part of the 'whisker' hash and which are part of the request/response hash, let's look at the possible options for the 'whisker' hash that relate to a HTTP packet.

Note that internal to the 'whisker' hash are the 'hin' and 'hout' hashes which are directly mapped to the request and response hash, respectively. For this article we will use the %request, %response and %jar PERL hashes to refer to the HTTP request hash, HTTP response hash and HTTP cookies hash, respectively.

1. `{'whisker'}->{'host'}` = This will be the remote host that we want to connect to. This value will also be reflected in the HTTP 'Host' header. This can be an IP address, DNS name or a NetBIOS name. If you want to specify another value in the HTTP 'Host" header then you will have to do it manually prior to using the 'LW2::http_do_request' function. The HTTP 'Host' header must be included in all HTTP 1.1 packets to be RFC compliant. By default this is 'localhost'.
2. `{'whisker'}->{'port'}` = This is the remote port that we want to connect to. The only