Example Recipe

8.8 Uploading Malicious File Contents

Problem

You want to test how your application handles files with malicious content. The content might be malicious because of its size, because it is not the required type, or because it actually causes the application to crash when it is processed.

Solution

Example 8-8. Uploading a file through Perl

```
#!/usr/bin/perl
use LWP::UserAgent;
use HTTP::Request::Common qw(POST);
$UA   = LWP::UserAgent->new();
$page = "http://www.example.com/upload.jsp";
$req = HTTP::Request::Common::POST( "$page",
   Content_Type => 'form-data',
   Content => [ myFile => [ 'C:\TEMP\myfile.pdf',
                            "AttackFile.pdf",
                            "Content-Type" => "application/pdf" ],
                Submit => 'Upload File',
              ]
   );
$resp = $UA->request($req);
```

Description

The code from Example 8-8 does the minimum possible work to upload a file named **`C:\TEMP\myfile.pdf`** (that lives on your local hard disk) and put it at the URL shown in the **`$page`** variable. It is clear from Example 8-8 that there are several opportunities for malicious attack.

The first obvious thing to try when testing for security this way is to provide contents of files that will cause difficulties at the server. If the requirements for your application say that files must be smaller than 100 kilobytes, your typical boundary case testing would involve uploading 105 kilobyte files, 99 kilobyte files, and probably 0-byte files.

You should also upload some extremely large files, too. A badly designed application might keep unacceptable files in some temporary location, even after it has sent a message to the user saying “file too large.” This means you could crash the application by filling its temporary storage, even though the files appear to be ignored.

From a security point of view, good tests will send files whose contents are not what they appear. Imagine a web application that unpacks uploaded ZIP files, for example. You could take a file like a spreadsheet or an executable, rename it to end in **`.zip`**, and then upload it. This would surely cause a failure of some kind in your application. Some file formats have old, well-known attacks. For zip files there are attacks called “zip bombs” or “zip of death” attacks where a correctly formatted zip file that is very small (for example, 42 kilobytes) would expand to over 4 gigabytes if fully unzipped. You can find an example file by searching on Google for "zip of death."

Other data formats have similar possible bugs. It is possible to craft various image files that contain size information indicating that they are one size (e.g., 6 megabytes) but actually only contain a fraction of that data—or much more than that data.