ARTICLES

HTTP Authentication

January 1998 / Web Project / HTTP Authentication

Web servers implement basic authentication in many ways. If they don't do what you need, create your own scheme.

Jon Udell

The user/password dialog box (see the figure) is familiar to every Web user. You encounter it when you try to fetch a URL that is protected. If you type in a user name and password that the Web server will accept, the request will retry and succeed. Otherwise, it will retry and fail, with a message such as "Authorization Required."

This simple scheme has a surprising number of permutations. Web servers, for example, can declare protected zones in very different ways. You can protect scripts as well as documents. There are all sorts of ways to manage lists or databases of users, passwords, and groups.

This month, I'll review the authentication features of my two favorite Web servers -- Internet Information Server (IIS) and Apache. But first, by way of background, I'll show how I added HTTP authentication to ByteCal, my servlet-based group calendar (see http://www.byte.com/art/9708/sec8/art1.htm).

Authentication with ByteCal

If you run ByteCal under the Java Web Server (JWS), which supports HTTP authentication, you can protect it as you protect any other resource. Just run the JWS server-administration applet and associate ByteCal with a list of authorized users. But what if you run ByteCal under Acme.Serve, a Java-based Web server that does not support HTTP authentication? To explore how basic authentication works, I added authentication to ByteCal, so that the servlet itself, independently of any protection that its host may or may not provide, can authenticate users.

Here's how it works. When you issue a request to ByteCal, it checks the HTTP headers sent by the browser, looking for one called Authorization. If it's absent, the requesting browser has not yet authenticated itself to ByteCal. So, ByteCal issues a challenge in the form of this pair of HTTP headers:

```
HTTP/1.0 401 Unauthorized
WWW-Authenticate:
  Basic realm="ByteCal"
```

The crucial part of the first header is the 401 code; that's what provokes authentication. Although I'm declaring the protocol to be HTTP/1.0, HTTP/0.9 or HTTP/1.1 should work identically -- basic authentication is the same in all cases.

The second header defines the type of authentication that will occur. In theory, this could be basic or digest. In practice, it's almost always basic, a weak protocol that sends credentials as clear text. The digest method, which encrypts the credentials and is implemented in some servers (Apache, JWS), is unfortunately not yet supported by either major browser (see the sidebar "Digest Authentication"). For the exclusive combination of IIS server and Windows-based Microsoft Internet Explorer (MSIE) clients, there's a third option (see the sidebar "NTLM Authentication").

The final piece of the WWW-Authenticate header is the realm, in this example ByteCal. It distinguishes this protected zone from others that might possibly be in effect on the same server.

When the browser receives the authentication headers, it displays its user/password dialog box to the user. When the user fills in the fields and clicks OK, the browser retries the request and tacks on an Authorization header. Here's the example given in the HTTP specification for user "Aladdin" with password "open sesame":

```
Authorization: Basic
  QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

This mangled representation of "Aladdin:open sesame" appears to be encrypted, but actually it's not. It's only MIME-encoded (Multipurpose Internet Mail Extensions), aka base 64-encoded. Routines to decode the credentials string are available for Java, Perl, and many other languages.

Let's recap. ByteCal looks for an authorization header. If it's absent, ByteCal issues a basic authentication challenge to the browser. The browser prompts the user for a name/password combo. Then the browser sends a MIME-encoded representation of these back to ByteCal in the form of an authorization header.

ByteCal decodes the authorization header and decides whether or not to grant access. For now, it's a simple match against a single name/password hard-coded into ByteCal on behalf of a group of users. The check occurs at the top of ByteCal's main service routine. If it succeeds, ByteCal dispatches the appropriate handler for the request. If it fails, the service routine prints an "Authorization Failed" message and then returns immediately. What if it did not? I made that mistake. The result: A user could simply bypass authentication by canceling out of the dialog box.

ByteCal looks for the environment variable HTTP_AUTHORIZATION, not for HTTP_REMOTE_USER, which is how Web servers normally pass the name of an authenticated user to a script. Why? HTTP_REMOTE_USER isn't one of the headers that a browser sends to a server. An authenticating Web server adds HTTP_REMOTE_USER when it invokes a script that's been accessed by way of an authentication protocol. (Some servers also subtract HTTP_AUTHORIZATION -- more on this later.)

But recall that Acme.Serve is not an authenticating Web server. Thus, ByteCal sees different headers than a typical protected Common Gateway Interface (CGI) script sees: HTTP_AUTHORIZATION is present, but HTTP_REMOTE_USER is absent.

Note that ByteCal is now in a position to do some fancy things. It could, for example, deny everyone write access to user A's calendar except A and A's assistant, B. Basic authentication as typically implemented in Web servers can't offer you this flexibility. The reason is not that the protocol precludes it, but rather that the usual URL-oriented protections don't map to arbitrary application data. When an application bypasses the Web server's authentication mechanism and supplies its own, it can provide such a mapping.

Apache Authentication

Apache supports two ways to protect directories from which you serve content or run scripts. You can use the <Directory> directive in the server configuration file (or in VirtualHost sections within that file), or you can use .htaccess files located in the directories they protect. The .htaccess method is more flexible but slower. It lets you adjust security policy on the fly but requires the server to reread the .htaccess file for each request. The <Directory> method is less flexible but faster. You have to restart the server to adjust policy, but there's no per-request overhead.

Where does Apache look up users and groups? There are all sorts of options. Here's an .htaccess file that refers to a text file containing user names and encrypted passwords:

```
AuthType Basic
AuthName OurUsers
AuthUserFile /plain/ourusers
require valid-user
```

The file /plain/ourusers, created using the htpasswd command, is the Web analog to a Unix /etc/passwd file. If you're handling thousands of users, you probably don't want Apache to have to read a huge password file every time it authenticates. So, the same .htaccess file could instead look like this:

```
AuthType Basic
AuthName OurUsers
AuthDBMUserFile /dbm/ourusers
```

Now when a user authenticating to the realm OurUsers sends a name and password, Apache looks up the credentials not in a text file, but in a much faster DBM database (disk-based hash). The file /dbm/ourusers can be created using a Perl script called dbmmanage that comes with Apache. To use the DBM method, you'll need to edit Apache's Configuration file, activate the relevant module (mod_auth_dbm), and rebuild Apache.

It gets even better. With mod_perl, Doug MacEachern's implementation of in-process Perl for Apache, you can write your own authentication module in Perl. Here's how it works. At each stage in the processing of a request, Apache calls a handler. These are usually written in C and linked with Apache (IIS users: Think "Internet Server [ISAPI] filter"). In the case of an AuthUserName directive, the handler is Apache's built-in authentication module. For AuthDBMUserFile, it's mod_auth_dbm. But if you have installed mod_perl and your

.htaccess file looks like this:

```
AuthType Basic
AuthName OurUsers
PerlAuthenHandler Apache::Anon
```

Apache will call the Perl module Anon.pm. MacEachern wrote this module just to illustrate the concept of an Apache/Perl module. (IIS users: Think "ISAPI filter written in Perl," a lovely concept that sadly isn't yet possible with IIS and Win32 ISAPI Perl.) Anon.pm approves only requests from user name "anonymous."

But the point is that any Perl code can run in this context. A Perl-based authentication module can examine and modify Apache's internal request structure and use any algorithm and any Perl-accessible data source to decide whether to grant access.

Note that such a module has complete access to the HTTP headers sent by the client. If you write a CGI script to enforce a security policy, à la the ByteCal example above, that script will normally see only the user's name (HTTP_REMOTE_USER) and not the full credentials (HTTP_AUTHORIZATION).

That's because Apache, as a security measure, withholds the Authorization header from CGI scripts. (If you really want to build a CGI-based access-control script, you can tweak Apache to make it send this header.) But an Apache/Perl authentication module, running inside the server, knows everything that Apache knows about a request.

Authentication with IIS

IIS unifies Web-server security and native NT file-system security. Is this a feature or a bug? It depends. For intranet servers, it's a feature. You've already defined users and groups, and assigned file-system permissions accordingly, so why not leverage that infrastructure when building Web-server-based applications? There's also another advantage. With Apache and other Unix Web servers, there's no easy way to achieve file-level protection. Because IIS integrates with the NT file system, it's as easy to protect an individual file as it is to protect a whole directory.

For Internet servers, though, IIS's integrated security looks more like a bug. IIS itself can run only as a valid NT user. Internet clients become that user when they connect anonymously to IIS. To protect content or scripts, you revoke that anonymous user's rights to some directory. When a browser requests something in that directory, IIS issues a basic authentication challenge. (Alternatively, it can issue an NTLM challenge; see the sidebar "NTLM Authentication.")

What credentials will work here? The user name and password of any valid NT user who is listed in the local or domain accounts database and who has appropriate read or (in the case of a script) execute rights in that directory.

The problem with this scheme is that any accounts that you create for this purpose are meaningful not only to the Web server, but more broadly to the NT machine or even its entire domain. A rogue script running under such an account could be very dangerous.

What's more, if you use basic authentication, you're sending in clear text the name and password of a real NT account.

Out of the box, IIS offers no good way to handle the authentication of thousands of users on a public Internet-connected NT box. Clearly, you're not going to create thousands of local or domain accounts to handle this situation. You'll need to write an ISAPI filter that intercepts and handles the SF_NOTIFY_AUTHENTICATION event or acquire one that does this -- for example, Philippe Tenenhaus's Dynamic Authentication Filter (which is found at http://daf.simplenet.com/).

As you can see from this, basic authentication itself is a simple protocol. However, Web servers implement it in different ways, and those implementations govern what you can and can't do with the protocol. If you run into a roadblock, you'll have to modify your Web server, adding a module or filter that replaces the built-in authentication mechanism. Otherwise, you can bypass the Web server entirely and create your own authenticating application.

TOOLWATCH

IAIK Java Cryptography Extension
Internet: http://kopernikus.iaik.tu-graz.ac.at/IAIK-JCE/index.html

This amazingly complete crypto toolkit for Java reimplements and extends Sun's own JCE. There's support for X.509 certificates, multiple ciphers, stream- and block-oriented encryption, and more. The compiled code is free for noncommercial use. A licensed version ranges from $100 to $5000 depending on terms and options.

BOOKNOTE

SGML and HTML Explained....................$39.95
by Martin Bryan
Addison-Wesley
ISBN: 0-201-40394-3
Internet: http://www.aw.com/devpress

All the recent XML buzz is bound to create new interest in how SGML Document Type Definitions are constructed and how SGML relates to HTML. Here's the reference you'll need.

SGML and HTML Explained

photo_link (43 Kbytes) 

You Must Authenticate to Pass the Box

screen_link (15 Kbytes) 

The familiar user/password dialog box.

Jon Udell is BYTE's executive editor for new media. You can reach him at jon@byte.com.
