Handling Web Logins

By David Adams

Technical Note 06-39

Overview

The 4th Dimension Web server provides support for HTTP passwords through the **On Web Authentication** database method. Once activated and configured, this feature automatically sends a challenge whenever a protected method or other database resource is requested. However, there is a limitation to this system. If the password window is cancelled, a blank screen is displayed in the browser. This blank screen is confusing for end users and makes it look like the Web site has disappeared. Instead of a blank screen, it is far better to show a page with help text, login hints, links to other pages on the site, or any other relevant information. Overcoming **On Web Authentication**'s default behavior is not difficult, but it does require using a different strategy. Instead of using the automatic system, authentication is performed with a few lines of custom code in **On Web Connection** and any methods called through `4DACTION`.

This technical note explains why **On Web Authentication** system works as it does and describes the replacement Web password system. Included with this note are two sample databases, Web_Login_Default, which illustrates how **On Web Authentication** behaves, and Web_Login_Custom, which implements the custom solution.

About Web Passwords

To understand **On Web Authentication**'s behavior, it's necessary to explain how HTTP passwords work. First, consider how password protected Web sites typically behave when the password dialog is cancelled. As a user, here's what seems to happen:

1. A page is requested in the browser.
2. The server asks for a user name and password in a dialog.
3. If the password dialog is cancelled, the server sends back an error page.

Internally, the actual steps are different:

1. A page is requested in the browser.
2. The server sends an HTTP status code of 401 and an error page.
3. The browser understands the 401 status code as an indication that a user name and password are required.

4. The browser presents a dialog to collect the user name and password. However, the browser does not display the error page, at this point.
5. If the user cancels the password dialog, the browser displays the error page returned by the server in the second step.

In case the differences between the apparent and actual steps are unclear, the significant details are emphasized below:

- The password dialog is displayed and controlled by the browser, not the server. Therefore, different browsers draw different looking dialogs.
- If the password dialog displayed by the browser is cancelled, **there is no message sent to the Web server**. Therefore, the error page must already be present in the browser.

With these points in mind, let's look at how 4th Dimension works.

4th Dimension's Default Behavior

The **On Web Authentication** database method is included in every 4th Dimension database as a place to include Web authentication code. The method runs whenever a URL or semi-dynamic callback invokes a 4D method. **On Web Authentication** automatically receives six text parameters with various pieces of information related to the Web request. To accept a request, return **True** in $0. To send a password challenge to the browser, return **False** in $0. The imaginary code below shows how all of this works:

```
C_BOOLEAN($0;$acceptConnection_b)
C_TEXT($1)    `  ;$url_t)
C_TEXT  ($2)    `;$httpHeader_t)
C_TEXT  ($3)    `;$clientIPAddress_t)
C_TEXT  ($4)    `;$serverIPAddress_t)
C_TEXT  ($5;$userName_t)
C_TEXT  ($6;$password_t)

$userName_t:=$5
$password_t:=$6

If (WebLoginIsOkay ($userName_t;$password_t)
   $acceptConnection_b:=True
Else
   $acceptConnection_:=False
End if

$0:=$acceptConnection_b
```

What happens when $0 is set to **False**? 4th Dimension terminates processing the request and returns an HTTP response with the `401 Unauthorized` status. Additionally, it includes the `WWW-Authenticate` HTTP header required for a complete HTTP password challenge response. The HTTP response below shows exactly what the headers include:

```
HTTP/1.1 401 Authorization Required.
Server: 4D_WebStar_D/2004
Date: Tue, 10 Oct 2006 00:21:20 GMT
WWW-Authenticate: Basic realm="Web_Login_Default.4DB"
```

A response with a status of 401 *may* include a page of content to display if the password dialog is cancelled, but the response generated by **On Web Authentication** includes nothing. Therefore, when a password dialog is cancelled, there is nothing to display. There is no way to manually add an error page with the response sent by **On Web Authentication**. The way around this limitation is to accept all requests in **On Web Authentication** and then handle authentication manually in **On Web Connection** and/or in methods called by `4DACTION`.

Implementing a Custom Web Password Challenge

Overview

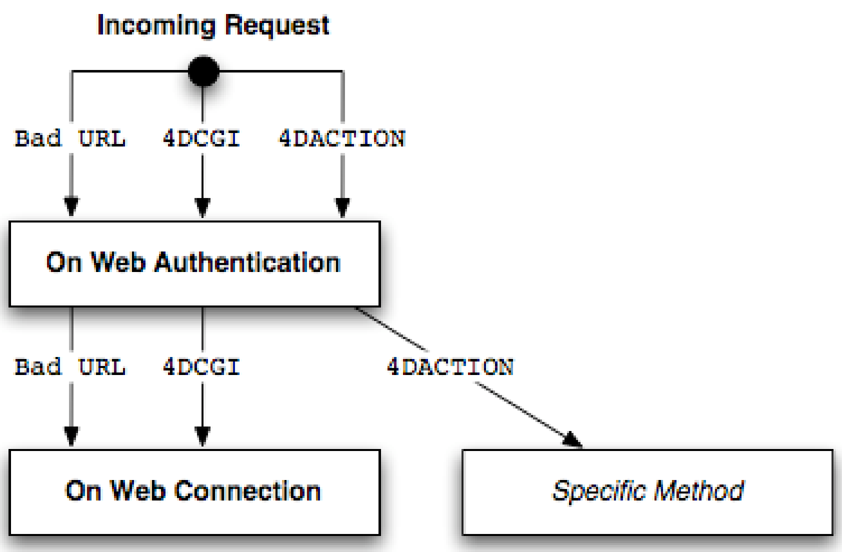
Implementing a custom Web password challenge system in 4th Dimension is pretty simple. There are only two important tasks involved:

1. Test for passwords, when needed.
2. Send a complete, custom password challenge (HTTP 401) response, when needed.

Depending on how the database is used over the Web, there are a few ways to approach these tasks. The strategy implemented in the example database and discussed in this technical note supports password protecting all non-contextual access, including requests for individual pages, `4DCGI` calls, and `4DACTION` calls.

4D Web Server Flow of Control

Internally, the flow of control within the Web server varies slightly amongst the different non-contextual calling systems, as illustrated in the diagram below:



Notice that all three request types provoke **On Web Authentication** and that `4DACTION` calls bypass **On Web Connection**. Therefore, the most obvious place to put authentication code is in **On Web Authentication**. However, it's **On Web Authentication**'s limitations that lead us to build a custom solution in the first place. If all calls are channeled through **On Web Connection**, then authentication can be performed there. However, if some methods are called through `4DACTION`, they must handle authentication individually. Fortunately, none of these steps are hard and all of them are implemented in the Web_Login_Custom database. We'll look at the code needed in each location, next.

Customizing On Web Authentication

In the custom Web password solution, **On Web Authentication** requires a bit of code to provisionally accept the request and to store the incoming user name and password for later testing, as shown below (the code in the sample database is somewhat expanded):

```
C_BOOLEAN($0;$acceptConnection_b)
C_TEXT($1;$2;$3;$4;$5;$6)

` Save user name and password in process variables for 4DACTION methods.
C_TEXT(WebDemo_UserName_t)
C_TEXT(WebDemo_UserPassword_t)
WebDemo_UserName_t:=$5
WebDemo_UserPassword_t:=$6

$0:=True
```

Returning **True**, as shown above, bypasses 4th Dimension's automatic password behavior. Additionally, it allows every Web request through. Therefore, it is important to test all requests by hand. There are only two places where testing needs to be done, as mentioned: in **On Web Connection**, and in each method called directly by `4DACTION`. For sites that use `4DCGI` instead of `4DACTION`, all security testing can be handled in **On Web Connection**.

Tip `4DACTION` offers no meaningful advantages over `4DCGI`. in fact, it brings several disadvantages, for example, actual 4th Dimension method names being exposed through the Web. Apart from any security consideration, a corollary of this behavior is that changing the method name in 4th Dimension breaks all external links.

Customizing On Web Connection

The **On Web Connection** method runs each time a bad URL or `4DCGI` call are submitted. Below is the code used in the Web_Login_Custom database:

```
C_TEXT($1;$url_t)
C_TEXT  ($2)`;$httpHeader_t)
C_TEXT  ($3)`;$clientIPAddress_t)
C_TEXT  ($4)`;$serverIPAddress_t)
C_TEXT  ($5;$userName_t)
C_TEXT  ($6;$password_t)

$url_t:=$1
$userName_t:=$5
$password_t:=$6

If (WebLoginIsRequired ($url_t))
    $loginOkay_b:=WebLoginIsOkay ($userName_t;$password_t)
    If (Not($loginOkay_b))
      WebLoginSendChallenge
    End  if
End  if

If ($loginOkay_b)

    Case of
      : ($url_t="/protected.html")
        SEND  HTML  FILE("protected/page_reached.html")

      : ($url_t="/4DCGI/CallMethodWith4DCGI@")
        MethodCalledBy4DCGI

      Else
        C_TEXT(WebDemo_RequestedURL_t)
        WebDemo_RequestedURL_t:=$1` Value is read semi-dynamically by the 404 page.
        SEND  HTML  FILE("not_found.html")
    End  case

End  if
```

All of the work comes down to three methods, *WebLoginIsRequired*, *WebLoginIsOkay* and *WebLoginSendChallenge*. None of these methods is long or complicated.

The WebLoginIsRequired Method

Web sites typically include a mixture of protected and public information. The *WebLoginIsRequired* method provides a central location for code to test if a specific URL requires authentication. Internally, this method uses some conventions to identify protected resources, as seen in the code below:

```
C_BOOLEAN($0;$loginRequired_b)
C_TEXT($1;$url_t)

$url_t:=$1
$loginRequired_b:=False

Case  of
   : ($url_t="/protected@")` Use a password on anything that is in the protected directory.
     $loginRequired_b:=True

   : ($url_t="/4DCGI@")` Use a password for all 4DCGI calls.
     $loginRequired_b:=True

   : ($url_t="/4DACTION@")` Use a password for all 4DACTION calls.
     $loginRequired_b:=True

   Else
     $loginRequired_b:=False
End  case

$0:=$loginRequired_b
```

The code from the Web_Login_Custom database, shown above, is little more than a skeleton. Within a specific database, the logic can be expanded, refined, and rewritten however is necessary.

The WebLoginIsOkay Method

The *WebLoginIsOkay* method in the sample database is effectively a stub. All it does is test if the user name and password provided over the Web match values hard-coded into the method itself. However, despite its simplicity, the method is correctly located in the call chain to handle all authentication tasks. The simplest form of the method is shown below:

```
C_BOOLEAN($0;$loginIsOkay_b)
C_TEXT($1;$userName_t)
C_TEXT($2;$password_t)

$loginIsOkay_b:=False

If  (($userName_t="guest") & ($password_t="4D"))
   $loginIsOkay_b:=True
End  if

$0:=$loginIsOkay_b
```

Within a production database, the logic for this method can be expanded, as needed. For example, the incoming values may need to be tested against user names and passwords stored in records or external documents.

The version of the method shown above is adequate, if `4DACTION` is not used. However, if `4DACTION` is used, the user name and password must be saved within **On Web Authentication** for inspection. There are several ways to code for this situation. The approach used in the sample database is to save the user name and password in **On Web Authentication**, as shown earlier, and then to test these values within *WebLoginIsOkay* when no parameters are provided. The expanded version of the method listed below is used in the sample database:

```
C_BOOLEAN($0;$loginIsOkay_b)
C_TEXT($1;$userName_t)
C_TEXT($2;$password_t)

$loginIsOkay_b:=False

If (Count  parameters=2)
    $userName_t:=$1
    $password_t:=$2
Else
    ` The user name and password are saved in On Web Authentication.
    If (Undefined(WebDemo_UserName_t))`   This shouldn't happen.
      WebDemo_UserName_t:=""
    End if
    If (Undefined(WebDemo_UserPassword_t))`   This shouldn't happen.
      WebDemo_UserPassword_t:=""
    End if

    $userName_t:=WebDemo_UserName_t
    $password_t:=WebDemo_UserPassword_t

End if

If (($userName_t="guest") &( $password_t="4D"))
    $loginIsOkay_b:=True
End if

$0:=$loginIsOkay_b
```

This version of the method can now handle authentication requests from **On Web Connection** or any method called by `4DACTION`. Since the user name and password are already being stored by **On Web Authentication**, some developers may wish to simplify the method as follows:

```
C_BOOLEAN($0;$loginIsOkay_b)
C_TEXT($1;$userName_t)
C_TEXT($2;$password_t)

$loginIsOkay_b:=False

` The user name and password are saved in On Web Authentication.
If (Undefined(WebDemo_UserName_t))`   This shouldn't happen.
    WebDemo_UserName_t:=""
End if
```

```
If (Undefined(WebDemo_UserPassword_t))`  This shouldn't happen.
    WebDemo_UserPassword_t:=""
End if

If ((WebDemo_UserName_t="guest") & (WebDemo_UserPassword_t="4D"))
    $loginIsOkay_b:=True
End if

$0:=$loginIsOkay_b
```

The WebLoginSendChallenge Method

The code for the *WebLoginSendChallenge* may look a bit complex but, ultimately, it simply sets the HTTP header values needed to provoke an HTTP password challenge and adds the text of an error page. The complete code for the method is listed below:

```
ARRAY TEXT(WebDemo_HttpHeaderNames_at;3)
ARRAY TEXT(WebDemo_HttpHeaderValues_at;3)
WebDemo_HttpHeaderNames_at{1}:="X-VERSION"` This must be the first item in the array.
WebDemo_HttpHeaderValues_at{1}:="1.1"

WebDemo_HttpHeaderNames_at{2}:="X-STATUS"` This must be the second item in the array.
WebDemo_HttpHeaderValues_at{2}:="401"` Not Authorized. This signals the browser to ask for a
password.

C_TEXT($realm_text)
` This information is required for a password challenge,
` but  you can change the realm name to match your system.
$realm_text:=""` For example: Basic realm="Web Login Demo"
$realm_text:=$realm_text+"Basic realm="
$realm_text:=$realm_text+Char(Double quote )
$realm_text:=$realm_text+"Web Login Demo"
$realm_text:=$realm_text+Char(Double quote )

WebDemo_HttpHeaderNames_at{3}:="WWW-Authenticate"
WebDemo_HttpHeaderValues_at{3}:=$realm_text

SET HTTP HEADER(WebDemo_HttpHeaderNames_at;WebDemo_HttpHeaderValues_at)
SEND HTML FILE("password_challenge.html")
```

The HTTP/HTML capture listing below shows the output produced by the code shown above. The bulk of the output, starting from `<!DOCTYPE HTML`, are the contents of the `password_challenge.html` document. The header sections modified by the code above are highlighted below for emphasis:

```
HTTP/1.1 401 Authorization Required.
Server: 4D_WebStar_D/2004
Date: Tue, 10 Oct 2006 02:28:11 GMT
WWW-Authenticate: Basic realm="Web Login Demo"
Connection: close
Last-Modified: Tue, 10 Oct 2006 02:28:11 GMT
Expires: Wed, 11 Oct 2006 02:28:11 GMT
Content-Type: text/html;Charset=ISO-8859-1
Content-Length: 1092
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
      <title>Password Required</title>

      <link
            rel="Stylesheet"
            type="text/css"
            title="Styles for 4D Web Log-in Examples"
            href="/styles.css"
            rev="Stylesheet">
</head>
<body>

<h1>User Name and Password Required</h1>

<p>A user name and password are required to access the requested page or
method. Enter a valid user name and password in the window provided by
the browser. An example of a browser password dialog is shown below:</p>

<img
      src="password_help.jpg"
      alt="Password dialog"
      width="527"
      height="385">

<p><b>Hint</b>: The user name is <span class="html_text">guest</span>
and the password is <span class="html_text">4D</span>. Remember, that
once a user name and password combination have been accepted, the
browser continues to send them along with every new request. Therefore,
once you enter the name and password listed above, all requests are
accepted until the browser has quit.</p>

<p><a href="/index.html">Home</p>

</body>
</html>
```

Authenticating from within 4DACTION Methods

The routines in the sample database are designed to support authentication from **On Web Connection** or an individual method. As the flow of control diagram listed earlier shows, calls to `4DACTION` pass through **On Web Authentication** and then to the specific method. In the custom database, the user name and password are saved in process variables known to the *WebLoginIsOkay* method; this approach simplifies authentication within specific methods, as illustrated in the example below:

```
C_TEXT($0;$1)`Required  for methods called through 4DACTION.

If (WebLoginIsOkay =False)
   WebLoginSendChallenge
Else ` Login good.
   SEND HTML FILE("protected/method_ran_through_4daction.html")
End if
```

On Web Authentication Versus the Custom Solution

The HTTP header output shown above is functionally identical to the headers produced by the On Web Authentication password challenge. The relevant sections of the two versions are shown side-by-side below for comparison:

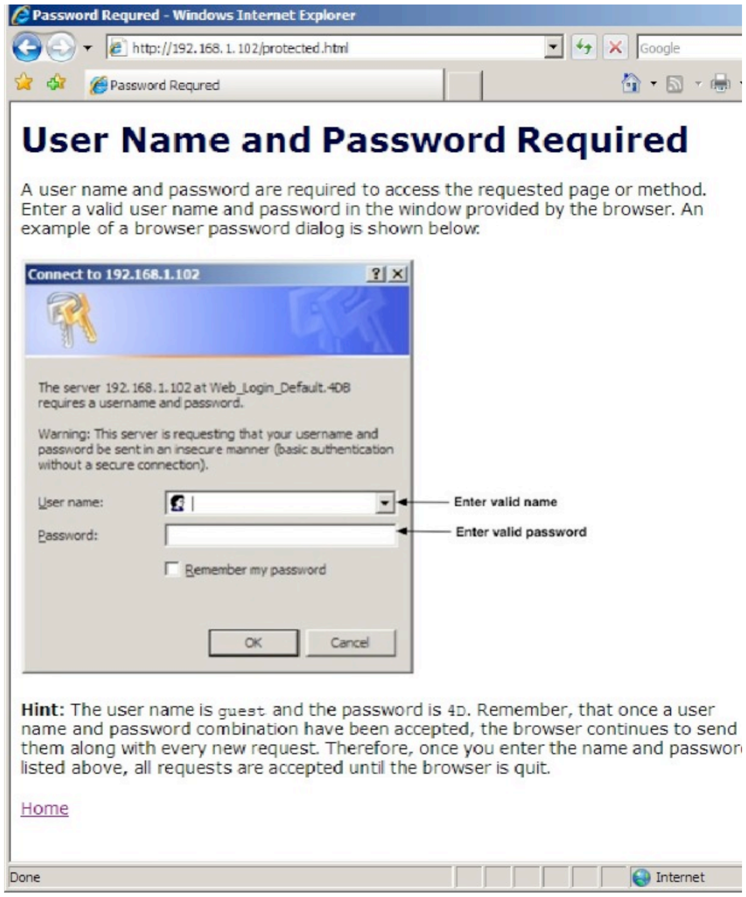
On Web Authentication

```
HTTP/1.1 401 Authorization Required.
Server: 4D_WebStar_D/2004
WWW-Authenticate: Basic
realm="Web_Login_Default.4DB"
```

Custom Solution

```
HTTP/1.1 401 Authorization Required.
Server: 4D_WebStar_D/2004
WWW-Authenticate: Basic realm="Web
Login Demo"
```

The only difference between the two is the name of the realm. In the custom solution, the realm name is freely configurable. In the **On Web Authentication** version, the realm name is automatically built from the database structure name. The meaningful difference between the two is what follows. In the case of **On Web Authentication**, nothing follows. With the custom solution, a full page of information follows. Then, if the password dialog is cancelled in the browser, the page included with the challenge is shown. Below is an illustration of the error page produced by the sample database:



Password Requred - Windows Internet Explorer
http://192.168.1.102/protected.html
Google
Password Requred

User Name and Password Required

A user name and password are required to access the requested page or method. Enter a valid user name and password in the window provided by the browser. An example of a browser password dialog is shown below:

Connect to 192.168.1.102

The server 192.168.1.102 at Web_Login_Default.4DB requires a username and password.

Warning: This server is requesting that your username and password be sent in an insecure manner (basic authentication without a secure connection).

User name:

Password:

Remember my password

Hint: The user name is `guest` and the password is `4D`. Remember, that once a user name and password combination have been accepted, the browser continues to send them along with every new request. Therefore, once you enter the name and passwor listed above, all requests are accepted until the browser is quit.

Home

Additional Notes on Web Passwords

Web Passwords Are Not Secure

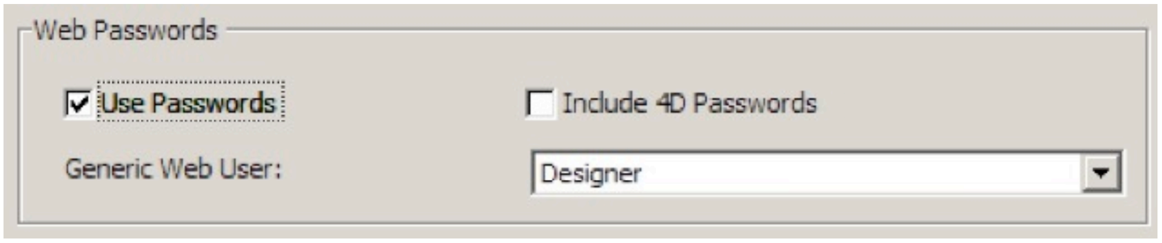
This following point cannot be overemphasized: **Web passwords are not secure**. This is not a bug, it's how the HTTP system was originally designed. When a user name and password are submitted, they are combined together and converted to base64. For example, the user name guest with the password 4D are transmitted as follows:

```
Authorization: Basic Z3Vlc3Q6NEQ=
```

The string `Z3Vlc3Q6NEQ=` is a base64 version of the string `guest:4D`. There is no encryption involved in HTTP Basic authentication at any point. Furthermore, once the password is accepted, any pages or other documents sent over the network are not secured. If a site needs encryption, use SSL (HTTPS).

Do Not Use 4D Passwords Over the Web

The 4th Dimension Web server password includes several options within the Database Preferences dialog on the Web > Advanced page, pictured below:



Web Passwords

[✓] Use Passwords

[ ] Include 4D Passwords

Generic Web User: Designer

If the Use Passwords option is selected, the Include 4D Passwords option may also be selected. In this case, 4th Dimension's automatic Web password system can compare incoming Web user names and passwords against 4D user names and passwords. Given that Web passwords are not secure, using this feature makes 4D passwords insecure. This situation is particularly dangerous in a mixed Web/4D Client environment. Exposing real 4th Dimension user names and passwords over the Web exposes credentials that can then be used to connect with 4D Client.

Note The Use Passwords option is required by the custom Web password system described in this technical note.

Browsers Remember Web Passwords

The Web is a stateless protocol, meaning no information is retained between requests. Therefore, if a Web site requires a password, the password must be sent in with each and every request. Yet, it's normal to enter a user name and password once at a site and then browse freely. To improve the user experience, the browser creates the illusion that a password is only required once. Internally, the browser remembers the user name and password and continues to send them with each new request, typically for as long as the browser remains open. If you're testing Web passwords, keep this point in mind. Sometimes, it's necessary to quit the browser and restart to continue testing. Another development strategy is to run multiple browsers to avoid having to quit one browser repeatedly.

Tip The indispensable Web Developer Firefox extension, available at http://chrispederick.com/work/webdeveloper/, can clear HTTP passwords on the fly. Look for the Miscellaneous > Clear Private Data > Clear HTTP Authentication option.

Case-Sensitivity Is Optional

It is up to each individual Web site to decide if user names and passwords should be handled case-sensitively or not. Within 4th Dimension, it is sometimes awkward to compare strings case-sensitively. For details on adding case-sensitivity to 4th Dimension projects, see 4D Technical Note 05-41, "Case-Sensitive Operations in 4th Dimension." The sample database includes a variety of read-to-use methods, including a case-sensitive string comparison routine named *CS_AlphasAreEqual*.

Summary

The 4th Dimension Web server provides automatic support for HTTP passwords through the **On Web Authentication** method. However, this feature doesn't support including an error page to display in case the password dialog is cancelled. Fortunately, this limitation is easily corrected by adding a short piece of custom code to **On Web Connection** and any method called directly through `4DACTION`.