Centralizing App Security Using Admin User Credentials

In many cases, developers offer a suite of BBj® business applications and include utility programs for the application administrators that setup and create a database connection or call BBjAPI().getAdmin() for system administrative functions such as looking at open files and registering BUI applications. Developers would also commonly recommend that their customers' administrators follow good security practices by changing their BBj Services password after installation, making it necessary for the utility programs to obtain the customer's usernames and passwords whenever needed.

You are likely one of these developers, so how do you go about making the administration program obtain this password and keep it secure, so that the program does not have to ask for username and password every time it is needed?

Today's Solutions

There are several ways to address this situation. One solution would be to hard code the client's password into the program and then compile the source into a protected program before distributing it to the client. However, this solution would cause some maintenance overhead for each client (either on the developer's part or the client's), not to mention the unfortunate side effect that anyone who can access the client would also be able to access the client's BBj Services.

A somewhat more secure alternative would be to create a properties file on each client's machine with the username and password stored in encrypted form. While possibly inconvenient, this solution requires that someone get access to the client's machine before accessing BBj Services.

Tomorrow's Solution, Today!

However, what if there was a way to keep persistent client credentials for accessing BBj Services, like a server-side cookie? Suppose then, that before a program attempts to make a call that requires user credentials, if it could not find a valid cookie for the client, it would present a login dialog, get the username and password, and any other configuration needed to access BBj Services. Upon a successful connection, the program would then optionally store the user credentials in a server-side cookie.

Sound great? Well, in the spirit of easing the developers burden and providing application building block tools, BASIS now provides this functionality in the Admin Utility!

Admin Utility Overview

The Admin Utility is located at <install_path>/utilities/admin.bbj and contains two BBj objects of relevance – Admin and UserCredentials. Admin is the object that asks for user credentials and creates the cookies; UserCredentials is obtained from the Admin object and contains the information necessary to connect to BBj Services. Look at how these objects handle user credentials:

1. Instantiate the Admin object.
2. Configure the Admin object (optional).
3. Before calling a section of code that requires credentials, obtain a UserCredentials object by calling Admin::getUserCredentials().
   a. The Admin utility will look for the user's cookie.
   b. If the Admin utility cannot find a cookie for the user, a login dialog box appears.
4. Use the fields in the UserCredentials object to connect to JDBC or BBj Services before performing the system administration tasks.

Run the simple program in **Figure 1**. If this is the first time you have ever run a program using the Admin utility and your BBj Services password is other than the default, the login dialog shown in **Figure 2** will display. Upon completing the fields and clicking [Login], the Utility verifies that you are able to connect to BBj Services. Your credentials are then saved to a cookie. Run this sample a second time and notice that since your credentials are saved, the dialog box does not display. >>

By Shaun Haney
Quality Assurance Engineer

```
rem Admin User Credentials Example: Obtain User Credentials to get a list of open files.

use ::admin.bbj::Admin
use ::admin.bbj::UserCredentials

declare Admin adminUtilityObject!
declare UserCredentials userCredentials!
declare BBjAdmin admin!
declare BBjVector openFileInfos!
declare BBjOpenFileInfo fileInfo!

REM We need to make a call to BBjAPI().getAdmin(), to obtain credentials
adminUtilityObject!=new Admin()
adminUtilityObject!.setRequiredPermissions(adminUtilityObject!.getDELETE_DB())
userCredentials!=adminUtilityObject!.getUserCredentials()

if userCredentials!=null() then
   print "Sorry, could not log in.  Press Enter.  ",
   read a$
   end
endif

userName$=userCredentials!.getUser()
password$=userCredentials!.getPassword()
admin!=BBjAPI().getAdmin(userName$,password$)
openFileInfos! = admin!.getOpenFileInfos()

numFiles=openFileInfos!.size()
if (numFiles<>0) then
   print "Currently open files:"
   for flCtr=0 to numFiles-1
      fileInfo!=cast(BBjOpenFileInfo,openFileInfos!.get(flCtr))
      print fileInfo!.getFilename()
   next flCtr
else
   print "No open files to report"
fi
```

Figure 1. Using the Admin Utility to get credentials

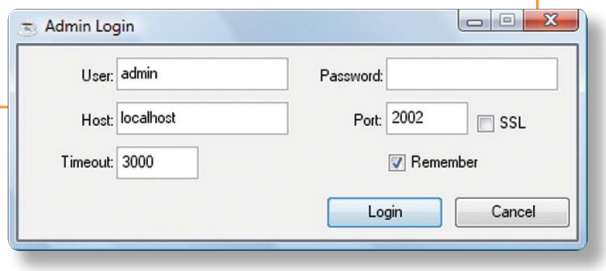

Figure 2. The resulting dialog from Figure 1 when no credentials are available

Admin Utility Options

Even though the Admin Utility has a small interface, its functionality is robust. Typically, username and password are the only user credentials developers consider. In addition to the required username and password, BBj Services also offers several configuration options for establishing the connection, and each database or administrative task requires privileges that the user may or may not have. Therefore, the Admin Utility demonstrates other user credentials to control both the connection to BBj Services and additional user privileges.

Pre-configure

The first feature is the ability to preconfigure BBj Services connection parameters and specify whether the user can modify them at login. Parameters that are configurable at the login prompt for BBj Services are the hostname, port, username, password, whether to use SSL, the timeout for obtaining a connection in seconds, and whether to remember these parameters on the next login. One can configure each of these parameters before showing the dialog via a simple "set" method. Additionally, one can configure them to be a "required" parameter or to set the field to the default value and prevent editing.

```
REM We need to make a call to BBjAPI().getAdmin(), to obtain credentials
adminUtilityObject!=new Admin()
adminUtilityObject!.setHostRequiredValue(1)
adminUtilityObject!.setPortRequiredValue(1)
adminUtilityObject!.setSSLRequiredValue(1)

userCredentials! = adminUtilityObject!.getUserCredentials()
```

Figure 3. Code from Figure 1 modified to make Host, Port, and SSL Uneditable

If a company wanted its employees to connect to a specific BBj Services, then it could pre-configure the host and port to point to its server according to the sample in **Figure 3**. When the login dialog pops up, the host and port would display in uneditable fields. The user's only choice would be to connect to the specified BBj Services. This feature is >>

useful in the common situation where users are only expected to enter a username and password, and should not change the other parameters.

Privileges

The next feature is the ability to specify which privileges the connecting user must have for a successful login. If the application has configured the Admin object to require certain user privileges, the user may specify the correct username and password, but still not be able to log in due to the user lacking one or more of the required privileges.

Imagine an administration utility that includes a tab for database configuration. Only users with the appropriate user credentials must be allowed to select the tab to configure the database. Mary, who uses this application, has a normal user account with the username "mary" for performing everyday tasks. She also has a second account with the username "mary_admin", for performing maintenance tasks that require special permissions.

Mary selects this tab and specifies her "mary" account in the user credentials. The "mary" account has permission to make a JDBC Connection for executing queries, but does not have permission to delete databases, which is one of the many permissions required to have access to the features in the tab. Even after Mary correctly enters her username and password, her login fails and she receives a message that she does not have the permissions required to switch to the tab.

After seeing the error message, Mary selects the tab again and now enters credentials for "mary_admin". Her administrative account has the privilege to maintain databases so she is now able to switch to the tab successfully and perform database administration.

```
REM We need to make a call to BBjAPI().getAdmin(), to obtain credentials
adminUtilityObject!=new Admin()
adminUtilityObject!.setRequiredPermissions(adminUtilityObject!.getDELETE_DB())
userCredentials!=adminUtilityObject!.getUserCredentials()
```

Figure 4. Privileges for login specified with the Admin::setRequiredPermissions()

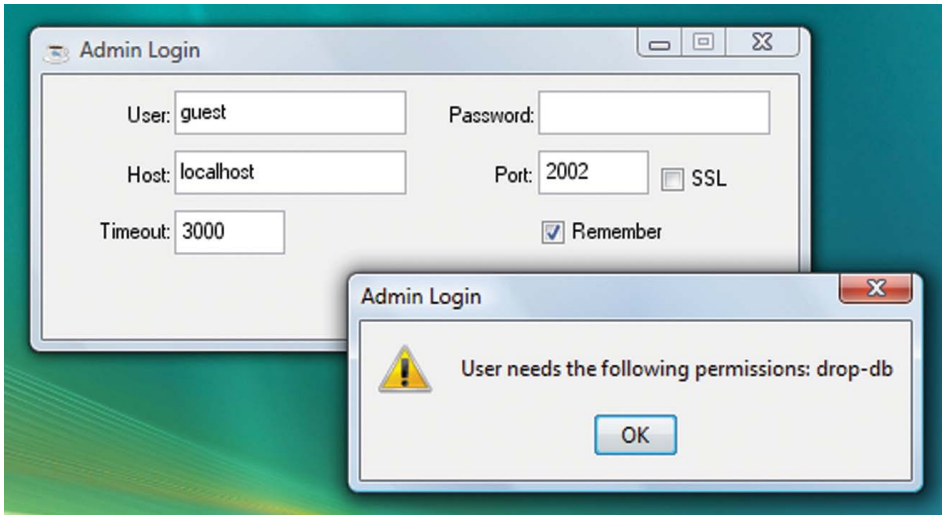

Figure 5. Log in attempt without privileges

Necessary privileges for login can be specified with the Admin::setRequiredPermissions() method (see **Figure 4**). **Figure 5** shows the resulting error message when "guest" attempts to log in without privileges. Each privilege is represented as a constant of BBjAdmin Object, not to be confused with the "Admin" Object from the Admin Utility. An example of one such BBjAdmin Object constant is BBjAdmin.DELETE_DB. Permissions are numeric constants and can be added together to set multiple permissions.

Summary

While requiring user credentials to access databases and administrative functions keeps clients secure, managing those credentials over multiple system administration programs can be inconvenient and potentially cause configuration and code maintenance nightmares if the solution is not carefully centralized. With the Admin Utility application building block, BASIS now centralizes that solution, along with all the configuration details needed to connect to BBj Services. Developers are now free to focus on developing applications without worry and unnecessary effort to secure their application. ■

Review the Admin Utility UserCredentials methods in the online documentation.