Simulating OPS$ Users through a Browser

Marc de Oliveira is Team Leader at Novo Nordisk Engineering. He has worked with Oracle products (mainly CASE/Designer) since 1989. He is coordinator of the Danish Designer SIG and the Scandinavian reseller of DesignAssist. Marc@deOliveira.dk.

What's your problem?

Why would you want to simulate OPS$ users through a browser? To those not familiar with OPS$ users I should quickly fill you in.

OPS$ users are also known as Operating System authenticated users. This is a very old Oracle feature that lets you bypass the Oracle login for users that have already been authenticated by a secure operating system like Unix or MS-network. In case we are sure that user X has been logged securely into the operating system there really is not much point in demanding an Oracle username and password, because we already know who the user is. So the OPS$ feature allows the end user to connect to an Oracle database without having to remember an Oracle username and password.

Sounds great, right? If all your users are authenticated when logging into the same MS-domain they should be able to pass that login information to the Oracle database automatically, so that they can run intranet applications without being prompted for an Oracle username and password.

Well, unfortunately Oracle does not support that, but if you would like to do it anyway then read on. The solution presented here is only relatively secure. It is secure enough to only allow users that are authenticated to the OS to login to the Oracle database. But smart Oracle users will be able to find a way to log into the database as another OS user than the one they logged into the OS as. You can do a number of things to diminish this loophole, but it will always be there. I'll get back to the security issues at the end of the paper.

The solution shown here will work on any Oracle WebServer (WAS, OAS or iAS). In the following I will be using the term WebServer referring to any of the Oracle WebServers.

Why simulate?

Why not use real OPS$ users? Operating System authentication has been a standard feature in Oracle for the past ten years but unfortunately it never made it into the Internet Age. "The Internet Changes Everything", well, for people using OPS$ users it really did, because they had to give up this feature to be able to let their users connect individually to the database through a browser (for WebForms, WebDB or Designer WSG modules).

OPS$ users still work in Client/Server mode for Forms, Reports, SQL*Plus etc but if the same users have to access Oracle through a browser as well, you must redefine them as regular Oracle users with an individual username and password, because OPS$ users and regular Oracle users cannot co-exist. You cannot create an Oracle user that can connect both as an OPS$ user (without the username and password) and as a regular Oracle user (with the username and password).

So, we need to define an architecture that will allow for Operating System authentication on both Client/Server and Web applications without using the build-in OPS$ feature. We have to simulate OPS$ users.

The Oracle username and password

The first thing to do is to define a way to map OS usernames with Oracle usernames and passwords.

On our site we have decided to map the OS username with the Oracle username directly but you are not restricted to that. You could decide to add OPS$ to the OS username, so that OS user USER1 logs into the database as OPS$USER1, or you could completely encrypt the OS username into an unrecognizable Oracle username, so that OS user USER1 becomes Oracle user XYZ123456.

The simplest possible function for mapping an OS username to an Oracle username follows here:

```
FUNCTION MAKE_USERNAME
   (p_os_username in varchar2)
  return varchar2 IS
   v_db_username varchar2(30);
BEGIN
   v_db_username:= p_os_username;
   RETURN (SUBSTR(v_db_username,1,30));
END;
```

In parallel, it is necessary to define a way to map the DB username with the Oracle database password. The following function does that in a very simple way. I will leave it up to you to improve the function to make a more advanced mapping.

```
FUNCTION MAKE_PASSWORD
   (p_db_username in varchar2)
  return varchar2 IS
   v_password number:= 1;
BEGIN
  FOR i IN 1..length(p_db_username) LOOP
   v_password:=
    v_password *
    ASCII(SUBSTR(lower(p_db_username),i,1));
  END LOOP;
  RETURN('PW'||SUBSTR(TO_CHAR(v_password),1,28));
END;
```

Creating database users

The next step is to create (or update) your database users using the above functions so that they all have a derivable password not known to themselves.

A create script would look something like this:

```
SET FEEDBACK OFF
SET LINESIZE 200
SET PAGESIZE 0
SET HEADING OFF
SPOOL create_user.ex
SELECT
 'CREATE USER '|| MAKE_USERNAME('&1') ||
```

```
 ' IDENTIFIED BY '||
 MAKE_PASSWORD(MAKE_USERNAME('&1')) ||
 ' DEFAULT TABLESPACE DEFAULT_TS '||
 ' TEMPORARY TABLESPACE TEMP_TS;'
FROM DUAL;
SPOOL OFF

START create_user.ex
HOST del create_user.ex
```

while a script for updating all your existing users would look something like this:

```
SET FEEDBACK OFF
SET LINESIZE 200
SET PAGESIZE 0
SET HEADING OFF
SPOOL update_users.ex
SELECT
 'ALTER USER '|| USERNAME ||
 ' IDENTIFIED BY '||
 MAKE_PASSWORD(USERNAME) ||
 ' DEFAULT TABLESPACE DEFAULT_TS '||
 ' TEMPORARY TABLESPACE TEMP_TS;'
FROM SYS.DBA_USERS
WHERE
 USERNAME IS NOT NULL AND
 USERNAME NOT IN ('SYS', 'SYSTEM');
SPOOL OFF

START update_users.ex
HOST del update_users.ex
```

Now that all your users do not know their own passwords you really need to go the next step, so that they can automatically be logged on!

Automatic login through the Web

The logging into the database through the web browser is divided into three steps. First, we need to extract the login information, then we need to perform the login without making the WebServer invoke the login dialog. The third step is a necessary extra redirection. These steps are explained in the following.

Extract the OS Username

The Oracle WebServer does not have direct access to the OS username of the users requesting access to the database, so we need to extract the OS username in the MS environment. This can be done easily by using a Microsoft product like Active Server Pages (ASP). The ASP page needs to be called with the name of the Oracle procedure to be executed as an argument, so that the user can be redirected to the desired PL/SQL module after login.

The URL to the ASP will look something like this (let us say that the ASP file is called START.ASP):

```
http://<URL to ASP directory>/
       start.asp?module=mymodule$.startup
```

To extract the argument "MODULE" and the current users OS username, the ASP file must look like this:

```
<%
p_module = Request("MODULE")
v_os_username =
 LCase(Request.ServerVariables("Remote_User"))
%>
```

We then need to make ASP functions corresponding to the above MAKE_USERNAME and MAKE_PASSWORD to map the extracted OS username to the corresponding database username and password. The main difference between ASP and PL/SQL is that you do not need to define your variables.

MAKE_USERNAME would look like this:

```
<%
v_db_username = v_os_username
%>
```

MAKE_PASSWORD would look like this:

```
<%
i = InStr(v_os_username,"\")
v_db_username =
 Mid(v_db_username,i+1,Len(v_db_username)-i)
v_password = 1
for j = 1 to Len(v_db_username)
 v_password =
  v_password*asc(Mid(v_db_username, j, 1))
next
v_password = "PW" & v_password
%>
```

At this point, the ASP page have values for p_module, v_os_username, v_db_username and v_password.

Note: If the module which you wish to call (p_module), has arguments you will have to hide the &-characters so that they are not interpreted as arguments for the ASP page. You could replace the &-characters with *-characters, like this:

```
http://<URL to ASP directory>/
       start.asp?module=mymodule$.startup?
       myarg1=A*myarg2=B
```

Log onto Oracle through the iAS

The actual logging into the Oracle database can be done from the above described ASP page using a little JavaScript to redirect the browser to an Oracle module. The login information can be placed in the URL like this:

```
http://<username>:<password>@<URL>
```

so the necessary JavaScript code will look as shown in figure 1.

The <URL to plsql-cartridge> looks different depen-

```
<%
response.write "One moment, please..."
Response.Write "<SCRIPT LANGUAGE=""JavaScript"">" + CHR(13)
Response.Write "<!--" + CHR(13)
Response.Write "{"+ CHR(13)
Response.Write "self.location.href = ""http://" & v_db_username & ":" & v_password &
"@<URL to plsql-cartridge>/redirect?p_arg1=" & p_module & """"+ CHR(13)
Response.Write "}"+ CHR(13)
Response.Write "// -->"+ CHR(13)
Response.Write "</SCRIPT>"+ CHR(13)
%>
```

Figure 1. Log onto Oracle through iAS

ding on which WebServer you are using.

The second redirection

Note that the above JavaScript is redirecting the browser to a plsql-module called REDIRECT with p_module as an argument. This second redirection is necessary to remove the username and password from the URL. If we did not do this, the username and password would permanently be visible in the URL, which would allow another user to note and reuse that username and password from his/her own computer to login as the other user.

Another problem with keeping the username and password in the URL is that List Of Values generated from Designer will not work because the URL somehow is not recognized.

As the REDIRCT procedure is called, the user is logged into the database, so we can now make the second redirection to the desired module without specifying the username and password.

The REDIRECT procedure should look something like this:

```
htp.print(
 '<html> '||
 '<head> '||
 '<META HTTP-EQUIV="REFRESH" CONTENT="0;'||
  ' URL='|| REPLACE(p_arg1,'*','&') ||
' ">'||
 '</head>'||
 '</html>');
```

The REPLACE is for converting the argument separation *-characters back to the necessary &-characters (se section Extract the OS Username).

Done!

With the above described ASP page and REDIRECT procedure installed on you system, MS-domain users will be able to access the Oracle procedure MYMODULE(MYARG1, MYARG2) through their own schema, without being prompted for a username and a password. Just let them click on a link like this from a simple HTML page:

```
http://<URL to ASP directory>/
        start.asp?module=mymodule$.startup?
        myarg1=A*myarg2=B
```

Automatic login through Client/Server

As Oracle schemes cannot be both OPS$ users and regular users at the same time we cannot use the supported OPS$ feature for our Client/-Server modules after having implemented the above architecture for connecting to the database through a browser.

Instead we need an architecture corresponding to the one for Web access where a startup form is called with a minimal connect string that can then calculate the database username and password to be used, make that connection and then call the desired form.

The Minimal Scheme

First you need a minimal scheme on the database that can connect to the database and nothing else. You might create the scheme like this:

```
CREATE USER STARTUP IDENTIFIED BY NOT_SECRET;
GRANT CONNECT TO STARTUP;
```

This scheme is used to launch the startup form with the name of the desired form as a parameter, like this:

```
Ifrun60.exe
  module=startup
  userid=startup/not_secret
  p_module=myform
```

The Logon Scheme

Next you need another minimal scheme on the database that can connect to the database and has access to the MAKE_USERNAME and MAKE_PASSWORD functions. You might create the scheme like this:

```
CREATE USER LOGON IDENTIFIED BY SECRET;
GRANT CONNECT TO LOGON;
GRANT EXECUTE ON MAKE_USERNAME TO LOGON;
GRANT EXECUTE ON MAKE_PASSWORD TO LOGON;
```

This scheme is used from within the startup form to connect to the correct database user and launch the desired form.

I need both the STARTUP scheme and the LOGON scheme to prevent end users from seeing the password of the LOGON scheme. More on this in the Security section.

The Startup Form

The startup form is a simple form with one small navigable item, a parameter called p_module and a when-new-form-instance trigger for launching the desired form.

The trigger will look like figure 2.

To get the OS username of the current user, I use the Win_API_Environment package contained in the D2KWUTIL.PLL library.

The newest version of the D2KWUTIL.PLL library can be downloaded from:

```
ftp://oracle-ftp.oracle.com/dev_tools/
```

```
declare
   v_os_username varchar2(30);
   v_db_username varchar2(30);
   v_password    varchar2(100);
   v_connect     varchar2(30);
begin
   v_os_username:= lower(Win_Api_Environment.Get_Windows_username);
   v_connect:=     get_application_property(connect_string);
   logout;
   logon('LOGON', 'SECRET@<connect string>');
   v_db_username:= make_username(v_os_username);
   v_password:=    make_password(v_db_username);
   logout;
   logon(v_db_username, v_password||'@'||v_connect);
   new_form(:parameter.p_module);
end;
```

Figure 2. When-new-form-instance trigger

```
patchsets/dev2k/Win95NT/d2kwutil/
```

That's it… The resulting functionality is that the end user starts up the startup form that barely shows up before the desired form is opened instead. The start-up form is not running in the background.

Security issues

In this section I will discuss some of the security issues raised by this OPS$ simulation.

The ASP files

The code for generating the database usernames and passwords is placed in the start.asp file which is just a simple text file. You should make sure that end users do not have read access to that file, while the ASP web server (Microsoft InformationServer) must be setup to be able to execute ASP files in the directory containing the start.asp file.

This can be achieved by placing the ASP file in a directory only accessible to one specific user (it could be the Administrator), and map that directory to a virtual directory defined to access the physical directory as that user. Make sure that the virtual directory is defined with the Source option disabled, so that end users cannot request to see the source code of the ASP file.

Another solution would be to wrap the code for making the username and password in an ActiveX component on the InformationServer and place the component somewhere not accessible to external users. Then the ASP file would just call the ActiveX component without revealing how the username and password is constructed.

Getting around the MS-domain

When logging into a PC, end users have the option to bypass the login dialog by pressing the Cancel button. When doing this, they will still have web access and, hence, be able to invoke the start.asp file.

If end users bypass the login dialog they will not get an OS username and therefore the start.asp program will not be able to log them into the database.

Even if the end user creates a local user and logs in locally as that user, that OS username will not be retrieved by the

```
Request.ServerVariables("Remote_User")
```

command.

The MAKE_USERNAME and MASK_PASSWORD functions

These functions can be used to generate the database username and password of any OS user so you should be very careful with whom you grant them to.

The only scheme that needs access to these functions is the LOGON scheme, so be very careful about who gets to know about the password to the LOGN scheme.

The password to the LOGON scheme is written in the STARTUP form, so it is important that end users do not have access to the STARTUP.FMB file.

The password to the STARTUP scheme is kind of public but that does not affect security as the START-UP scheme does not have access to anything.

The URL

As the database username and password is passed to the REDIRECT procedure through the URL, it will be visible to the end user for a short time while the page is performing the second redirection (see the section about the second redirection). This means that an end user is exposed to his/her own database username and password for a short time (less than a second) every time a start.asp link is invoked.

This is not a serious security problem because all users are able to connect as themselves, anyway.

It still makes it possible for one end user to see another end user's database username and password if he/she can get access to the other end user's PC. To prevent this problem, end users must know that they should not leave their PCs without locking them first (or log of the domain entirely).

Another solution is to expire the passwords often (see the next section).

Password expiration

Because all passwords are systematically generated using the function MAKE_PASSWORD it is easy to have them changed often to improve the security. By doing this, an end user that somehow got hold of another end user's database username and password (as explained in the previous section) would only be able to use that login information for a limited period.

If you make the MAKE_PASSWORD function dependent of the current date and have a recurring job updating all passwords accordingly every night just after midnight, any password would only be valid for 24 hours.

If you want to expire the passwords more often than this (every hour, maybe) you should make sure that the web server is set to "Keep Database Connection Open between Requests", so that users are not forced reconnect all the time.

Conclusion

So there you have it. If you trust your MS-network, with a little PL/SQL, JavaScript, ASP, HTTP and HTML it is possible to let your users individually log directly into the database without prompting them for at username and a password.

It is not a completely secure architecture but for most organizations I would say that it is secure enough because, 1) end users cannot get access to the database without logging into the MS-network first, and 2) after logging into the MS-network they have to be pretty smart to get access to another account than the one they logged onto the MS-network with.