Providing Password Protection to Excel Reports for Distribution in Emails

Applies to:

BusinessObjects Enterprise XI 3.1 SP2. For more information, visit the Business Objects homepage.

Summary

This paper is meant to provide a way to password protect the Excel output generated from a BusinessObjects (BO) report and then distribute it.

Author: Dhirendra Singh Khanka

Company: Tata Consultancy Services, Ltd.

Created on: 7 January 2011

Author Bio

Dhirendra has worked in the field of BusinessObjects reporting for the past two years.

Table of Contents

Introduction

This paper is meant to provide a way to password protect the Excel output generated from a BusinessObjects (BO) report and then distribute it.

It is done without using third-party add-ins to the BO installation. This method makes use of Visual Basic for Applications (VBA) for Excel and does not modify the BO installation by any means. The main objective is to provide another authentication level to the attachment sent by emails so that any accidental email sent to a non-intended person doesn't compromise the confidential data.

The Process

The idea is to create an executable (.exe) or a batch file (.bat) in the BO server installation machine which takes care of the task of automating the password protection of Excel outputs generated by BO.

Consider a case where some reports need to be scheduled for refresh and then output Excels to be sent to respective users of these reports. First a folder should be created specifically for this project on a Windows platform accessible to the server directly or via FTP.

In this shared folder place three files mentioned below for processing these report Excel outputs. The executable file is a process which continuously monitors for presence of report Excel outputs. If found, it calls the macro. The macro then processes these Excel files. The macro quits when these files are processed, and then returns the control to the .exe which again continues monitoring this shared folder for new Excel files.

The Components of the Project

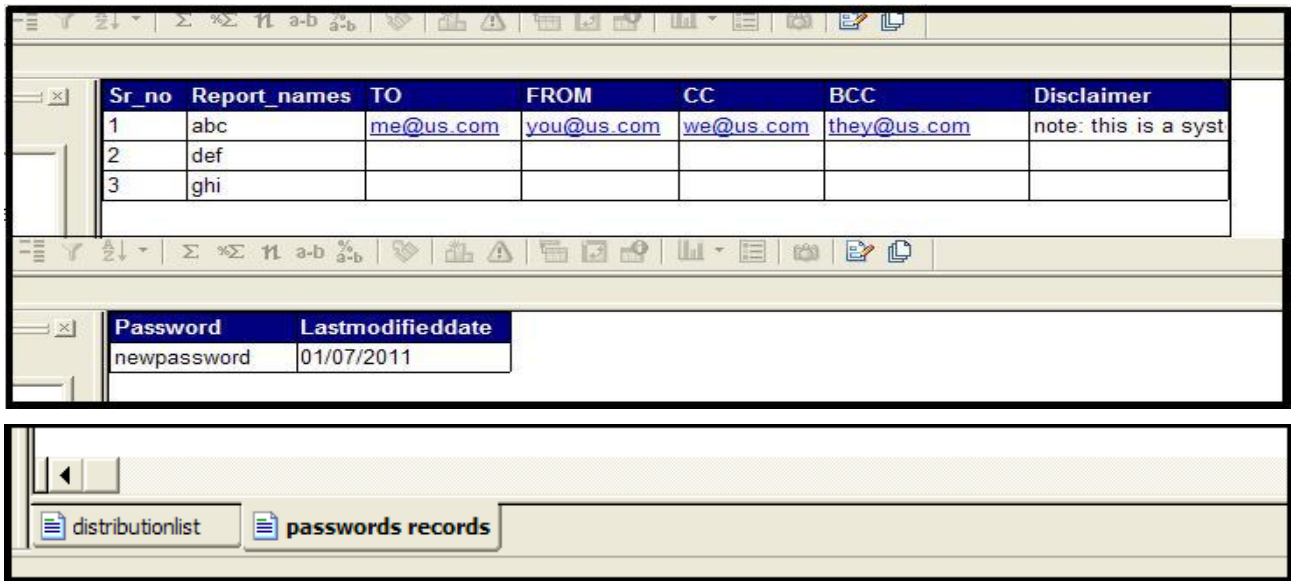
- Macro file (Excel)
- Report distribution list (Excel)
- Process (executable .exe/ batch file .bat)
- Scheduled task (to be created in windows operating system)

Report Distribution List

The report distribution list is an Excel file that contains the list of all the reports and their respective distribution email IDs. It also contains one tab showing the current password to be used for processing. This Excel file is generated using a BO report itself.

Whenever new reports need to be added to this project the report names should be updated in this particular report. Once done, this report needs to be scheduled and its Excel copy generated in the shared folder thereby replacing the old distribution list with new one.

A report is used directly here instead of using an Excel because direct access to a folder in the BO server machine may not be available. Below is the screen shot of the distribution list BO report.



Sr_no	Report_names	TO	FROM	CC	BCC	Disclaimer
1	abc	me@us.com	you@us.com	we@us.com	they@us.com	note: this is a syst
2	def					
3	ghi					

Password	Lastmodifieddate
newpassword	01/07/2011

Note: The tab names name should match with those mentioned in the get_rep_dist_data2 subroutine in the macro.

Excel macro

The Excel macro contains a VBA macro that processes the remaining Excel files excluding the distribution list and the macro Excel itself. Below is the code explaining all the subroutines.

Create a module and name it “Auto_Open” as this will create a subroutine that makes the code run automatically when this workbook is opened.

Write all the code within this subroutine. This macro checks for the files dumped in the shared folder after successful schedule and refresh.

It picks up one file at a time. If a match is found in the distribution list then it processes it and sends it to email subroutine, else it will just delete that original file and the processed file if a match was not found. This ensures no file is saved in the shared folder after processing; only a copy will be present in InfoView.

Even if an Excel report was mistakenly placed, then it will be processed and both of them deleted once a match is not found in the DL (distribution list).

```
Public Errstatus As Boolean
Sub Auto_Open()
'Auto_Open Macro
'Macro recorded 12/23/2010 by 5hirendra
 ‘Application.WindowState = xlMinimized
‘Application.ScreenUpdating = False      ‘uncomment this prior deployment
‘Application.Visible = False             ‘uncomment this prior deployment
‘Application.DisplayAlerts = False       ‘uncomment this prior deployment
Errstatus = False
‘Workbook_Open                           'uncomment this prior deployment
End Sub
Private Sub Workbook_Open()
On Error GoTo ErrorHandler:
 Dim reportdir As String
 Dim wb As Workbook
 Dim PathName As String
 Dim pathname2 As String
 Dim temppath As String
 Dim temppath1 As String
 Dim length As Integer
 Dim madte As String
 Dim repfolder As String
 Dim formdate As String
 Dim filename As String
 Dim filename2 As String
 Dim reportlocation As String
 Dim reportlocation2 As String
 Dim repcount As Integer
 Dim reparray() As String
 Dim filecheck As Integer
 Dim returnedarray() As String
 Dim i As Integer
 Dim uubound As Integer
 Dim mdate2 As String
 Dim mdate3 As String
 Dim formdate2 As String
 Dim temp As String
 Dim attachmentdate As String
 Dim rep_distribution_list As String
formdate = Replace(FormatDateTime(Now, vbLongTime), ":", " ")
```

```
formdate2 = Replace(FormatDateTime(Now, vbLongTime), ":", "-")
mdate = Format(CVar(Date), "MmmddYYYY ")
mdate2 = Format(CVar(Date), " mm-dd-YY ")
mdate3 = Format(CVar(Date), " YYYY-MM-DD ") & formdate2
attachmentdate = mdate2 & " " & formdate2
PathName = Application.ActiveWorkbook.Path & "\"
pathname2 = ThisWorkbook.Path & "\"
repfolder = PathName & mdate & formdate
rep_distribution_list = PathName & "ZZZ_rep_distribution_list.xls"
'for testing error handling you can temporarily rename the DL name above

'the report is named as ZZZ_ because I want the this file to be checked at the last
'from a group of Excel files. I am assuming none of the reports start with ZZZ
    If Dir(repfolder, vbDirectory) = "" Then MkDir repfolder
'create and empty temp directory for processing and saving with timestamp
'creation of new file is necessary as it has to be saved with timestamp
    reportdir = Dir(PathName & "*.xls")
'this statement picks up a name of an Excel file alphabetically from a folder
    filename = Replace(reportdir, ".xls", "")
    filename2 = filename & attachmentdate & ".xls"
    returnedarray = get_rep_dist_data2(rep_distribution_list)
'call is made to a function defined below which stores the DL Excel
'data in a 2 'dimensional array

    uubound = UBound(returnedarray) - 1
'this returns the number of reports present in the list
  ' ********************** main process***********************************
Do While reportdir <> "ZZZ_rep_distribution_list.xls"
 If PathName & reportdir <> ActiveWorkbook.FullName Then
    Set wb = Application.Workbooks.Open(PathName & reportdir, True, True)
    temppath = wb.Name
        reportlocation = repfolder & "\" & temppath
    reportlocation2 = repfolder & "\" & filename2
    wb.SaveAs filename:=reportlocation2, Password:=returnedarray(1, 8)
    wb.Close
  For filecheck = 1 To uubound
   If filename = returnedarray(filecheck, 2) Then
  'Call Send_Email_Using_CDO(reportlocation2, returnedarray, filecheck, mdate3)
'Uncomment this prior deployment, it is the subroutine which sends emails
'carrying the attachment which is password protected
'mention of call keyword here is mandatory

  Exit For
   End If
  Next filecheck

   Kill (reportlocation2)
   Kill (PathName & reportdir)
End If
   reportdir = Dir
Loop
   RmDir (repfolder)

'deletes the new password protected file after sending of the mail
'deletes the original file in the storage folder once processed as cleanup part
'points to any Excel file in storage location alphabetically
```

```
'deletes the empty folder which was created,
'after the processing of the entire batch
'****************************** main process end**********************
If Errstatus = False Then
Application.Quit
End If

ErrorHandler:
If Err.Number <> 0 And Errstatus = False Then
'if error not captured before then log the error
Errlog = FreeFile
Open pathname2 & "Errorlog" & attachmentdate & ".txt" For Output As #Errlog
Write #Errlog, Err.Description
Close #Errlog
Err.Clear
ElseIf Err.Number <> 0 And Errstatus = True Then
' if error already logged then remain in idle state till the code is debugged
End If
End Sub
' this error handler is for error in mailing subroutine
Public Sub errorhandling()
Dim pathname2 As String
Dim mdate2 As String
Dim formdate2 As String
mdate2 = Format(CVar(Date), " mm-dd-YY ")
formdate2 = Replace(FormatDateTime(Now, vbLongTime), ":", "-")
attachmentdate = mdate2 & " " & formdate2
pathname2 = ThisWorkbook.Path & "\"
Errlog = FreeFile
Open pathname2 & "Errorlog" & attachmentdate & ".txt" For Output As #Errlog
Write #Errlog, Err.Number & Err.Description
Close #Errlog
End Sub

' This error handler is for DL capture function
Public Sub errorhandling2()
Dim pathname2 As String
Dim mdate2 As String
Dim formdate2 As String
mdate2 = Format(CVar(Date), " mm-dd-YY ")
formdate2 = Replace(FormatDateTime(Now, vbLongTime), ":", "-")
attachmentdate = mdate2 & " " & formdate2
pathname2 = ThisWorkbook.Path & "\"

Errlog = FreeFile
Open pathname2 & "Errorlog" & attachmentdate & ".txt" For Output As #Errlog
Write #Errlog, Err.Number & Err.Description
Close #Errlog
End Sub
'This function returns the DL data in the form of a 2-D array.
'It is important to 'make it as a 2-D array so that function call is reduced to just
'one.
'And any data can be accessed from the returned array without need to call function
'again.
Private Function get_rep_dist_data2(its_path As String) As Variant
On Error GoTo ErrorHandler:
```

```
Dim wb1 As Workbook
Dim sh1 As Sheets
Dim sh2 As Sheets
Dim to_names() As String
Dim cc_names() As String
Dim bcc_names() As String
Dim from_names() As String
Dim new_password As String
Dim rep_names() As String
Dim rep_count As Integer
Dim colmn_count As Integer
Dim row_num As Integer
Dim all_param() As String
Set wb1 = Application.Workbooks.Open(its_path, True)

With wb1.Worksheets("distributionlist").UsedRange.Rows.Count
rep_count = wb1.Worksheets("distributionlist").UsedRange.Rows.Count
ReDim all_param(1 To rep_count, 1 To 8)

For row_num = 2 To rep_count
all_param(row_num - 1, 1) = rep_count  ' total count
all_param(row_num - 1, 2) = Cells(row_num, 2) ' Rep names
all_param(row_num - 1, 3) = Cells(row_num, 3) ' TO
all_param(row_num - 1, 4) = Cells(row_num, 4) ' FROM
all_param(row_num - 1, 5) = Cells(row_num, 5) ' CC
all_param(row_num - 1, 6) = Cells(row_num, 6) ' BCC
all_param(row_num - 1, 7) = Cells(row_num, 7) 'Disclaimer
all_param(row_num - 1, 8) = wb1.Worksheets("passwords records").Cells(2, 1)
' password
  Next row_num
End With

get_rep_dist_data2 = all_param
'return the two dimensional array which is a snapshot of the report distribution
‘list

wb1.Close False
ErrorHandler:
If Err.Number <> 0 Then
Errstatus = True
' set the error capture flag
Call errorhandling2
End If
End Function

'Sending a text email using authentication against a remote SMTP server
Private Sub Send_Email_Using_CDO(wblocation As String, reparray() As String, rownum
As Integer, repdate As String)
On Error GoTo ErrorHandler:
Const cdoSendUsingPickup = 1
Const cdoSendUsingPort = 2
Const cdoAnonymous = 0
' Use basic (clear-text) authentication.
Const cdoBasic = 1
' Use NTLM authentication
Const cdoNTLM = 2 'NTLM
```

```
Dim objmessage As Object
' Create the message object.
Set objmessage = CreateObject("CDO.Message")
objmessage.From = reparray(rownum, 4)
' Set the TO Address separate multiple address with a comma
objmessage.To = reparray(rownum, 3)
objmessage.cc = reparray(rownum, 5)
objmessage.BCC = reparray(rownum, 6)
objmessage.Subject = reparray(rownum, 2) & " Refreshed As of " & repdate
objmessage.TextBody = _
"Hi," & _
vbCrLf & _
"Please find attached " & reparray(rownum, 2) & _
" refreshed on " & repdate & vbCrLf & _
 vbCrLf & reparray(rownum, 7)
' Or you could use HTML as:
' objMessage.HTMLBody = strHTML

'ATTACHMENT : Add an attachment Can be any valid url
'objmessage.AddAttachment ("file://D:\BO docs\temp\temp\Backup\t1.xls")
objmessage.AddAttachment (wblocation)
'This section provides the configuration information for the SMTP server.
'Specifie the method used to send messages.
objmessage.configuration.fields.Item _
("http://schemas.microsoft.com/cdo/configuration/sendusing") = _
cdoSendUsingPort
'The name (DNS) or IP address of the machine
'hosting the SMTP service through which
'messages are to be sent.
objmessage.configuration.fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpserver") = _
"abcd.xyz.com"
' Specify the authentication mechanism to use.
objmessage.configuration.fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate") = _
cdoAnonymous
'cdoBasic
' we will use anonymous authentication as this wont require storing of a SMTP
'username and password ‘inside the macro.
'However a valid email has to be present in the from field, any invalid email or a
'string entered shows a error ‘while sending the mail. I am not sure if there is a
'way to pass a string instead of valid email id, so you can ‘keep the report owners
'email id in the from field.

' The username for authenticating to an SMTP server
'using basic (clear-text) authentication, not required for cdoanonymous
'objmessage.configuration.fields.Item _
'("http://schemas.microsoft.com/cdo/configuration/sendusername") = _
'"abc@xyz.com"

'The password used to authenticate, not required for cdoanonymous
'to an SMTP server using authentication
'**************************
'objmessage.configuration.fields.Item _
'("http://schemas.microsoft.com/cdo/configuration/sendpassword") = _
'"empty not required"
```

```
'The port on which the SMTP service
'specified by the smtpserver field is
'listening for connections (typically 25)
'You can check your SMTP Server on SMTP port 25 with the following Telnet command:
'Open a command line and type
'telnet smtp-server.domain.com 25

objmessage.configuration.fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpserverport") = _
25

'Use SSL for the connection (False or True)
objmessage.configuration.fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpusessl") = _
False

' Set the number of seconds to wait for a valid socket to be established with the
‘SMTP service before timing out.
objmessage.configuration.fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpconnectiontimeout") = _
60

' Update configuration
objmessage.configuration.fields.Update
' Use to show the message.
' Send the message.
objmessage.Send

'cleanup
Set objmessage = Nothing
'Set objmessage.configuration = Nothing
'Set objmessage.configuration.fields = Nothing

ErrorHandler:
If Err.Number <> 0 Then
Errstatus = True
' set the error capture flag
Call errorhandling
End If
End Sub


'I have tried to include as many comments as possible, so that it is easy for any
'one who has no knowledge of VBA but is aware of basic programming constructs.

‘*****************************END OF
CODE*****************************************
```

Executable or Batch File?

Once the macro file is ready, build a program that can trigger this macro when there are Excels to be processed. Although a batch file can do the trick, what is preferable is an executable file without a window that runs in memory. This ensures that the application is not accidentally closed. A batch file on other hand can at maximum only be minimized on start, but cannot be made to run without a window.

Below is a batch file converted into an executable file.

```
@echo off
REM d:
REM cd Business Objects\testing\testing 1jan
REM set timer for the batch file to run as approx 6 hours
REM set /a timer = 20000
:start
 setlocal enableextensions
set /a count = 0
echo %timer%
echo %count%
for %%x in (*.xls) do set /a count+=1
 echo %count%
if Not %count% == 2 test.xls
REM test is the Excel file containing the macro, the count should not be different
REM than 2, i.e. Macro file and DL
 endlocal
  @ping 127.0.0.1 -n 500
REM set /a timer = timer - 500
REM echo %timer%
REM if %timer% == 0  goto :end
goto :start
```

Place the above code in a text file using a text editor such as Notepad. Save it with a **.bat** extension. Normally, when a .bat file is clicked, it will execute the instructions in it and exit when finished. What is wanted in this case is that the process shouldn't exit and it should continuously monitor the shared folder for new Excels. In doing so, the window will never close. Therefore, there is a need for an .exe.

Lines starting with "REM" call for a comment. It is recommended that comments are included to make the code more readable. Whenever any command is typed at the **cmd** prompt, it gets displayed so that you know what you entered. In case of a batch file this is not desired because it is meant to perform its job and then exit.

This will echo off all the commands so that they do not appear on the screen. There is an option to explicitly mention the drive where the macro is placed. Otherwise, comment them if there is uncertainty where it will be placed during deployment. While designing the .exe file, make sure to keep the running directory parameter as the current directory of the .exe file and not temp directory because then the .exe will not be able to locate the macro file.

If a continuous process is not used, then make use of a timer to stop it and an **appworx** module to trigger it. However, if an .exe is made, then there needs to be a way to terminate the main .exe. The batch file using cmd.exe (a child process) will stop, but the main process will continue to running in background. Instead, make a continuous process. It won't use much of the RAM, and there is no need to monitor these checks.

Introduce a wait time of about 10 minutes. Make use of the ping command, which pings the loop back address 500 times (approximately 10 minutes) before it can check for the count of Excels in the shared folder. Since Windows XP by default doesn't come packed with cmd utilities like **sleep**, **wait**, **pause**, use the ping command with this operating system.

Note the line containing (*.xls). Mention of the extension here is very important. If the .xls is missed and use only (*), the macro will get falsely triggered after every 10 minutes since the count is always more than two.

Scheduled Task

What happens to the process in case the machine running the process is shut down for maintenance? Here make use of the Windows scheduled task function found in the Control Panel to create a schedule. It may be necessary to have an administrator make the task for the process.

Check for the items below:

- Antivirus products may treat this file as suspicious as it came from an untrusted source and is not a digitally signed application.
- Create an everyday schedule to run this program every 15 minutes or so. Username and password must be provided to create a task else it gives an error.
- In the task settings, by default the option is selected to stop the task after running for 72 hours. It must be unchecked. Also all power management options need to be unchecked.
- In the Schedule Option Advanced settings, check the Repeat Task Option every 15 minutes. Give the duration as 23 hours.
- Check the option of "*if a task is still running. Stop it at this time*". This is the most important setting as it ensures that new instances of the same process are not started every 15 minutes. This will make sure that the process will get triggered once the machine restarts.

Some Considerations

- For the macro to run unattended, either the macro security on the server machine should be set to low, or a self-signed digital signature be created after placing the macro on the server machine. Next, the VBA code should be assigned this signature. Now a security alert won't pop for this particular Excel macro as it is digitally signed from the same machine.
- For the distribution list BO report, old passwords should be moved down the stack. The first row should always be the current password to be used to protect the Excels. Maintaining old password and their change date is a must as attachments can only be opened with passwords that were used when they were processed.
- The reports should be scheduled in the shared folder with their exact names and no timestamp should be appended to the Excel copies. Also, filenames should appear exactly with the same name as they are present in the repository, otherwise filename search in distribution list won't fetch anything and the file will simply be lost.
- Library references made in the VBA library (specifically the Microsoft CDO 1.21 library) should be present in the server machine for using the messaging system.
- Windows environment is required for running the macro and to store the report Excel copies.

Simple Advantages

- The process is fairly simple to implement without requiring third-party monitoring tools or user intervention.
- The VBA doesn't require any third-party DLL library.
- It can be left fully unattended as it is not depended any external trigger from an outside application.
- Files are permanently deleted from the shared folder, thereby leaving no confidential data behind for any possible misuse.
- For adding new files for distribution, only a BO report need to be modified, this can be safely kept in repository with password or write protection. There is no maintenance in the macro or .exe required at any time.

Related Content

Sending mail from Excel with CDO

3 Nifty Ways to Send Email Using VBA in Excel

Ozgrid forum thread: Send Email Using CDO

For more information, visit the Business Objects homepage

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.