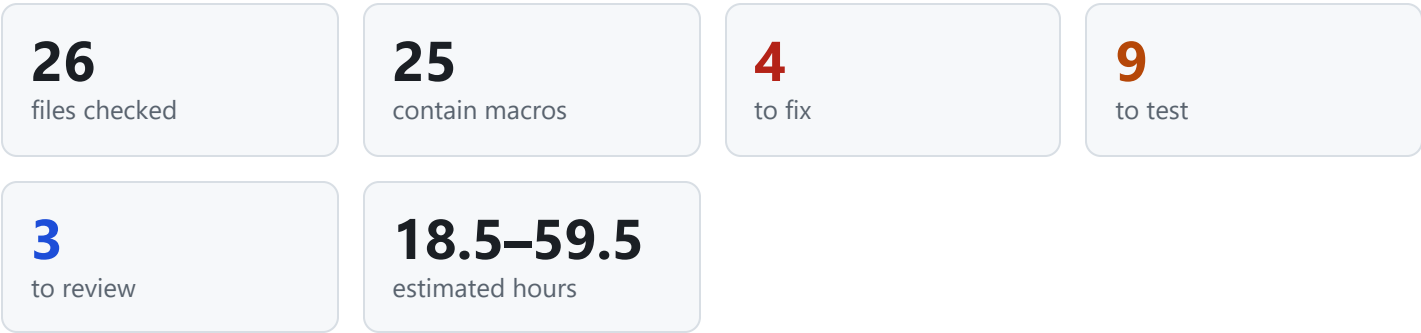


Macro check-up for Sample Co

October 23, 2026 · Prepared by Obegi Software LLC (Camas, WA)

Executive summary

We checked 26 files; 25 contain macros (4,912 lines of VBA). 4 files need fixing: the code shows they break on a current or announced Microsoft setup. 9 more files may break, depending on your Office build, settings or policies, and need a test. 3 more files have security items to review. Estimated work to fix and test: about 18.5–59.5 hours, plus 2–6 hours if users move to the new Outlook.



Risk tiers

Each file is in one tier: the most urgent thing found in it.

Tier	What it means	Files	% of files
Fix	The code shows something that breaks on a current or announced Microsoft setup.	4	15.4%
Test	May break, depending on your Office build, a setting or a policy. Test before relying on it.	9	34.6%
Review	Nothing breaks, but there is something a security team should look at.	3	11.5%
Plan	Works today, but is tied to a path, a program, a PC or classic Outlook.	6	23.1%
No action	Has macros, and nothing in them needs fixing, testing or review. (Facts such as "runs when opened" are still listed.)	3	11.5%
No macros	No macros in the file.	1	3.8%

What to do first

1. **P1 Fix Make the Windows API declarations 64-bit ready** 1–2 h
3 files · 4 places · 1 need testing · A VBA developer
Open each file in 64-bit Office, press Alt+F11 and run Debug > Compile. Add PtrSafe to each listed Declare and change handle and pointer arguments (hWnd, hDC, lpXxx) to LongPtr, then compile again and run the main macros once.
For example: File 0011.xlsm, File 0010.xlsm, File 0023.xlsm.
2. **P1 Fix Replace the VBScript files and the Script Control** 5.5–18 h
2 files · 6 places · 3 need testing · A VBA developer
Turn the VBScript feature off on one test PC (Settings > System > Optional features). Run each listed macro there; move what each .vbs script or Script Control call does into VBA or PowerShell.
For example: File 0009.xlsm, File 0022.xlsm.
3. **P2 Test Decide the ActiveX policy, or replace the controls** 2.5–10 h
3 files · 3 places · 3 need testing · IT admin, then a developer
Open one listed file on a current Microsoft 365 PC. If its buttons and boxes show as pictures, either allow ActiveX for trusted files by policy, or replace the controls with Form controls or shapes that run a macro.
For example: File 0004.xlsm, File 0016.xlsm, File 0024.xlsm.
4. **P2 Test Settle the Trust Center settings these files need** 2–7 h
3 files · 4 places · 4 need testing · IT admin
Decide, by policy, whether "Trust access to the VBA project object model" and Excel 4.0 macros are allowed for the users of these files. Where they are not, the listed macros need changing.
For example: File 0025.xlsm, File 0010.xlsm, File 0013.xlsm.
5. **P2 Test Confirm every PC is on Microsoft 365 Version 2508 or later** 0.5–1.5 h
2 files · 2 places · 2 need testing · IT admin
Check the Office version in the Microsoft 365 Apps admin center, or on a PC under File > Account > About Excel. On Version 2508 (build 19127.20154) or later these RegExp uses keep working; test the listed files on any older or perpetual Office.
For example: File 0014.xlsm, File 0018.xlsm.
6. **P2 Test Replace retired Office and Windows components** 1–2.5 h
2 files · 2 places · 2 need testing · A VBA developer (or IT admin, for .NET 3.5)
Run each listed macro on a freshly built PC. Where a component is missing, replace it as each finding says; for .NET Framework 3.5, decide whether to turn it on by policy instead.
For example: File 0017.xlsm, File 0024.xlsm.

7. **P2 Test Replace the Internet Explorer automation** 2–8 h
1 file · 1 place · 1 need testing · A VBA developer
Run each listed macro on a Windows 11 PC. Rewrite the web step with direct web requests (MSXML2.XMLHTTP or WinHttp) or the site's API; Edge cannot be driven the same way from VBA.
For example: File 0015.xlsm.
8. **P3 Review Review what these macros start, download, send and store** 3–8.5 h
6 files · 21 places · 19 need testing · Security or IT, with the file owner
For each listed file, have its owner confirm the programs it starts, the addresses it calls and the mail it sends. Change any password or key written in the code, then have the macro ask for it or read it from a protected store.
For example: File 0009.xlsm, File 0021.xlsm, File 0012.xlsm.
9. **P4 Plan List what these files depend on before moving shares or PCs** 1–2 h
5 files · 5 places · 5 need testing · IT admin, with the file owner
Add the listed paths, programs and add-ins to your migration plan. Move hard-coded paths into a settings cell, and confirm the listed software is on every PC that runs the file.
For example: File 0007.xlsm, File 0008.xlsm, File 0020.xlsm.
10. **P4 Plan Keep these users on classic Outlook, or move the email step** 2–6 h *only if users of these files move to the new Outlook*
1 file · 1 place · 1 need testing · IT admin, then a developer
Before switching anyone to the new Outlook for Windows, check whether they use a listed file. Either keep them on classic Outlook, or move the email step to Microsoft Graph or Power Automate.
For example: File 0003.xlsm.

The hours are a planning range, not a quote: typical times for each kind of change, applied to what the check found, for someone who knows VBA, including a basic test of each changed file. They do not cover rewriting business logic, code that could not be read, or waiting on a decision. The Outlook work (2–6 h) is only needed if users move to the new Outlook, so it is not in the total.

What only you can tell us

The answers decide most of the findings marked "Needs testing".

- Which Office version and update channel do your PCs run? Version 2508 or later settles the RegExp findings in 2 files.
- Is Office installed as 64-bit (the default since 2019) or 32-bit? The 64-bit findings in 5 files only break on 64-bit Office.
- Have you started turning VBScript off on PCs? 3 files depend on it.
- Do you allow ActiveX controls by policy, or keep Microsoft's default (off since April 2025)? It decides 3 files.

- Are the Microsoft Defender attack surface reduction rules for Office turned on (child processes, Win32 API calls)? They would block parts of 5 files.
- Do your policies allow "Trust access to the VBA project object model" or Excel 4.0 macros? It decides 3 files.
- Is .NET Framework 3.5 turned on in your PC build? 1 file needs it.
- Are you moving anyone to the new Outlook for Windows? 1 file drives classic Outlook, which the new one does not support.

Findings by kind

Will it break?

When	Files	Places in code
Breaks now	5	7
Will break when VBScript is disabled	3	6
Breaks on 64-bit Office	5	7
Unsupported, may stop working	1	1

"Needs testing" marks findings that depend on something the file cannot tell us (the Office build, a policy setting, what a variable holds when it runs). They are reported, not assumed safe. Of the 13 files above, 4 have at least one problem the code shows for certain.

Other things to know

Kind	Files	Places in code
Security-relevant	9	27
Maintenance risk	6	7
Info	1	1

- 2 files run code automatically when opened or closed.
- 1 file with macros is digitally signed; 24 are not. A "signed macros only" policy would block the unsigned ones.
- 1 file has a password-protected macro project; the code was still readable and was checked.
- 2 files contain ActiveX controls, which Microsoft 365 disables by default since 2025.
- 1 file contains Excel 4.0 (XLM) macro sheets.

In numbers

Files that...	Files	% of files with macros
Will break, or may break, on a current or planned setup	13	52%
Of which the code shows a certain break	4	16%

Files that...	Files	% of files with macros
Depend on VBScript (RegExp, Script Control, .vbs files)	4	16%
Have a Windows API declaration that will not compile in 64-bit Office	2	8%
Have anything that fails in 64-bit Office	5	20%
Automate or start Internet Explorer	1	4%
Have ActiveX controls in the document (off by default since 2025)	2	8%
Run code automatically when opened or closed	2	8%
Start other programs	2	8%
Call the Windows API	4	16%
Talk to the internet or send email	3	12%
Have a password or key written in the code	2	8%
Need software that Windows and Office do not include	3	12%
Drive classic Outlook (not supported by the new Outlook)	1	4%
Hard-code a file path (C:, a mapped drive, a server share)	2	8%
Read or change macro code at run time	2	8%
Need .NET Framework 3.5 (off by default in Windows 10/11)	1	4%
Have a password-protected macro project	1	4%
Are digitally signed	1	4%

File by file

Files with findings, most urgent first. Line numbers are as the VBA editor shows them (Alt+F11). Every finding is a row in `inventory.csv`, and every piece of work a row in `tickets.csv`.

1. File 0010.xlsm Fix

Excel macro-enabled workbook (.xlsm) · 9 modules, 274 lines of code · Not digitally signed · Estimated work: 1–2 h

Breaks now

Needs testing

Reads or changes macro code from a macro

func_debug, plugintimeExportCode, line 26

```
For Each ovbcomp In ThisWorkbook.VBProject.VBComponents
```

Why it matters. This code works with the VBA project itself. That needs the setting "Trust access to the VBA project object model", which is off by default and often locked off by policy. Without it this line fails with run-time error 1004. Code that writes code is also something security teams look for.

What to do. Confirm the setting is allowed for the users of this file, or remove the need to read or write code at run time.

Breaks on 64-bit Office

Windows API declaration without PtrSafe

f_uc_040_actionscamp, top of module, line 7

```
Private Declare Sub Sleep Lib "kernel32" (ByVal dwmilliseconds As Long)
```

Why it matters. In 64-bit Office (the default install since 2019) a Declare statement without the PtrSafe keyword is a compile error, so none of the macros in this file will run, not just this one.

What to do. Add PtrSafe to the Declare statement and change handle and pointer arguments (hWnd, hDC, lpXxx and the like) from Long to LongPtr. Keep the old line under #If VBA7 ... #Else if the file must still open in Office 2007 or older.

Security-relevant

Needs testing

Calls the Windows API (1 declaration: kernel32)

f_uc_040_actionscamp, top of module, line 7

```
Private Declare Sub Sleep Lib "kernel32" (ByVal dwmilliseconds As Long)
```

Why it matters. The macros call functions inside Windows directly (Declare statements). This is normal in older business macros, but it is also how malicious macros work. If your security baseline turns on the Microsoft Defender rule "Block Win32 API calls from Office macros", these calls are blocked.

What to do. Check whether your Defender attack surface reduction policy blocks Win32 API calls from Office macros. If it does, these macros need an exclusion or a rewrite.

Security-relevant

Needs testing

Can send email without Outlook (CDO)

f_uc_040_actionscamp, uc_a_Email, line 68

```
Set uc_CDOMail = CreateObject("CDO.Message")
```

 '创建对象

Why it matters. The macro creates a CDO email message, which can send mail straight to a mail server without Outlook. Security teams want to know about documents that send mail. (CDO is sometimes used only to read or decode messages; check what this code does with it.)

What to do. Confirm the mail server and credentials it uses; many Microsoft 365 tenants now block basic-authentication SMTP.

2. File 0009.xlsm Fix

Excel macro-enabled workbook (.xlsm) · 5 modules, 482 lines of code · Not digitally signed · 1 link to other workbooks · Estimated work: 3.5–11.5 h

Will break when VBScript is disabled

Runs a VBScript file

Module1, BACKGROUND, line 34

```
Shell "cmd.exe /c "" & PRGRM_DIR & "\POSTGRESQL_INSTALL_BACKGROUND.vbs"", vbHide
```

Why it matters. This line starts a VBScript (.vbs) file or the Windows Script Host. Microsoft says calling .vbs scripts from VBA will stop being supported as VBScript is removed from Windows; once VBScript is disabled (expected around 2026-27) the script will not run.

What to do. Move what the script does into the macro itself, or rewrite the script in PowerShell. Test now on a PC with the VBScript feature turned off.

Will break when VBScript is disabled

Needs testing

Refers to a VBScript file 3 places

Module1, Main, line 444

```
D = PRGRM_DIR & "\\execute\\" & TEMP_ID & "2_start.txt" & vbCrLf & "timeout 3 " & vbCrLf &
"" & PRGRM_DIR & "\\putty\\pscp.exe" -P ""&RC[-17]&" -pw ""&RC[-16]&"
root@""&RC[-18]&": ""if(RC[-1]="", "/root"", RC[-1])&" /PG_bkbs...
```

Module1, Main, line 452

```
Print #4, """" & PRGRM_DIR & "\log\" & HTNM & OS_type(iii) & ".vbs""
```

Module1, Main, line 465

```
Filename8 = PRGRM_DIR & "\log\" & TEMP_ID & ".vbs"
```

Why it matters. The code names a .vbs file. If the macro writes or runs that script, that part stops working once Windows disables VBScript (Microsoft expects that around 2026-27).

What to do. Check what the macro does with the file, and move the script's logic into VBA or PowerShell.

Security-relevant

Needs testing

Calls the Windows API (2 declarations: kernel32)

Module1, top of module, line 1

```
Public Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal Milliseconds As LongPtr)
```

Why it matters. The macros call functions inside Windows directly (Declare statements). This is normal in older business macros, but it is also how malicious macros work. If your security baseline turns on the Microsoft Defender rule "Block Win32 API calls from Office macros", these calls are blocked.

What to do. Check whether your Defender attack surface reduction policy blocks Win32 API calls from Office macros. If it does, these macros need an exclusion or a rewrite.

Security-relevant

Needs testing

Starts another program 9 places

Module1, BACKGROUND, line 34

```
Shell "cmd.exe /c "" & PRGRM_DIR & "\\POSTGRESQL_INSTALL_BACKGROUND.vbs"", vbHide
```

Module1, download, line 59

```
Shell "cmd.exe /c "" & PRGRM_DIR & "\\execute\\mklist.bat""
```

Module1, download, line 133

```
Shell "cmd.exe /c "" & PRGRM_DIR & "\\execute\\mklist.bat""
```

Module1, Main, line 150

```
Shell "cmd.exe /c copy /Y "" & PRGRM_DIR & "\\putty\\putty64bit\\*"" "" & PRGRM_DIR & "\\putty\\""
```

Module1, Main, line 152

```
Shell "cmd.exe /c copy /Y "" & PRGRM_DIR & "\\putty\\putty32bit\\*"" "" & PRGRM_DIR & "\\putty\\""
```

Module1, Main, line 411

```
Shell ("" & Filename2 & ""), vbHide
```

Module1, Main, line 460

```
Shell "" & Filename4 & ""
```

Module1, Main, line 462

```
Shell "" & Filename4 & "", vbHide
```

Module1, Main, line 476

```
Shell "cmd.exe /c "" & Filename8 & "", vbHide
```

Why it matters. This line starts a program or opens a file with another program. Security teams review this, and it only works where that program exists. If your security baseline turns on the Microsoft Defender rule "Block all Office applications from creating child processes", this line is blocked.

What to do. Confirm the program is present on every PC and allowed by your application-control policy; if the Defender rule is on, agree an exclusion or move this step out of the macro.

3. File 0022.xlsm **Fix**

Excel macro-enabled workbook (.xlsm) · 9 modules, 157 lines of code · Not digitally signed · Estimated work: 3–7.5 h

Breaks on 64-bit Office **Uses the Microsoft Script Control** 2 places

ThisWorkbook, Update, line 82

```
Set sc = CreateObject("ScriptControl"): sc.Language = "JScript"
```

ThisWorkbook, Testit, line 161

```
Set sc = CreateObject("ScriptControl"): sc.Language = "JScript"
```

Why it matters. The Script Control (msscript.ocx) only exists in 32-bit form, so this fails in 64-bit Office, the default install since 2019. Where it runs VBScript code (rather than JScript), it also depends on the VBScript engine Windows is retiring.

What to do. Replace it with plain VBA (for reading JSON, a VBA JSON module does the same job), or keep this file on 32-bit Office only.

Security-relevant **Talks to a web server** 2 places

ThisWorkbook, Update, line 74

```
Set request = New ServerXMLHTTP60
```

ThisWorkbook, Testit, line 155

```
Set request = New ServerXMLHTTP60
```

Why it matters. The macro makes web requests. This is common for pulling rates or data, but it is also how malicious macros download payloads, so security teams want to know.

What to do. Confirm the addresses it calls are expected, and that your web filtering allows them.

Info **Schedules a macro to run later**

ThisWorkbook, Timer, line 41

```
Application.OnTime Now + TimeValue("00:05:00"), "ThisWorkbook.Timer"
```

Why it matters. Application.OnTime makes a macro run at a set time, without anyone clicking anything.

What to do. Know which macro it schedules and why.

4. File 0011.xlsm **Fix**

Excel macro-enabled workbook (.xlsm) · 8 modules, 599 lines of code · Not digitally signed · Estimated work: 0.5–1 h

Breaks on 64-bit Office

Windows API declaration without PtrSafe 2 places

Main, top of module, line 1

```
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
```

OperationKey, top of module, line 1

```
Private Declare Function GetAsyncKeyState Lib "User32.dll" (ByVal vKey As Long) As Integer
```

Why it matters. In 64-bit Office (the default install since 2019) a Declare statement without the PtrSafe keyword is a compile error, so none of the macros in this file will run, not just this one.

What to do. Add PtrSafe to the Declare statement and change handle and pointer arguments (hWnd, hDC, lpXxx and the like) from Long to LongPtr. Keep the old line under #If VBA7 ... #Else if the file must still open in Office 2007 or older.

Security-relevant

Needs testing

Calls the Windows API (2 declarations: kernel32, user32)

Main, top of module, line 1

```
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
```

Why it matters. The macros call functions inside Windows directly (Declare statements). This is normal in older business macros, but it is also how malicious macros work. If your security baseline turns on the Microsoft Defender rule "Block Win32 API calls from Office macros", these calls are blocked.

What to do. Check whether your Defender attack surface reduction policy blocks Win32 API calls from Office macros. If it does, these macros need an exclusion or a rewrite.

5. File 0024.xlsm Test

Excel macro-enabled workbook (.xlsm) · 6 modules, 64 lines of code · Not digitally signed · Estimated work: 0.5–3 h

Breaks now

Needs testing

12 ActiveX controls placed in the document

Why it matters. The file contains ActiveX controls (buttons, boxes, lists placed on the sheet or page). Since April 2025 Microsoft 365 disables ActiveX controls by default (Office 2024 since October 2024): they show as pictures and cannot be used, unless an admin or the user changes the setting.

What to do. Replace them with Form controls or shapes that have a macro assigned, or allow ActiveX for trusted files by policy (Trust Center > ActiveX Settings).

Breaks now

Needs testing

Uses a .NET Framework object

Module1, testIBPVBA, line 51

```
Set memberDimensionary = CreateObject("System.Collections.ArrayList")
```

Why it matters. This .NET Framework class is only available to VBA when the Windows feature ".NET Framework 3.5 (includes .NET 2.0 and 3.0)" is turned on. It is not on by default in Windows 10 or 11, so on a fresh PC this line fails with an automation error.

What to do. Turn on .NET Framework 3.5 on every PC that runs the file (by policy or in Windows Features), or replace the object with a VBA Collection, a Dictionary or plain VBA.

6. File 0025.xlsm **Test**

Excel macro-enabled workbook (.xlsm) · 4 modules, 88 lines of code · Not digitally signed · Estimated work: 0.5–1 h

Breaks now

Needs testing

Reads or changes macro code from a macro 2 places

ExportVBAMain, mainExportVBACode, line 19

```
Set srcVBAProject = srcWb.vbProject
```

UDF, f_isComponentCodeModuleEmpty, line 59

```
If vbaComponent.CodeModule.CountOfLines < 3 Then f_isComponentCodeModuleEmpty = True
```

Why it matters. This code works with the VBA project itself. That needs the setting "Trust access to the VBA project object model", which is off by default and often locked off by policy. Without it this line fails with run-time error 1004. Code that writes code is also something security teams look for.

What to do. Confirm the setting is allowed for the users of this file, or remove the need to read or write code at run time.

7. File 0013.xlsm **Test**

Excel macro-enabled workbook (.xlsm) · 4 modules, 40 lines of code · Not digitally signed · Estimated work: 1–4 h

Breaks now

Needs testing

Contains 1 Excel 4.0 (XLM) macro sheet

Why it matters. The workbook has Excel 4.0 macro sheets, a macro language from 1992. Microsoft 365 disables XLM macros by default (since 2022) because attackers used them, so these macros do not run unless an admin allows them.

What to do. Rebuild the XLM macros in VBA, or confirm they are no longer used and delete the macro sheets.

8. File 0016.xlsm Test

Excel macro-enabled workbook (.xlsm) · 6 modules, 12 lines of code · Not digitally signed · Estimated work: 0.5–1.5 h

Breaks now

Needs testing

2 ActiveX controls placed in the document

Why it matters. The file contains ActiveX controls (buttons, boxes, lists placed on the sheet or page). Since April 2025 Microsoft 365 disables ActiveX controls by default (Office 2024 since October 2024): they show as pictures and cannot be used, unless an admin or the user changes the setting.

What to do. Replace them with Form controls or shapes that have a macro assigned, or allow ActiveX for trusted files by policy (Trust Center > ActiveX Settings).

9. File 0014.xlsm Test

Excel macro-enabled workbook (.xlsm) · 3 modules, 80 lines of code · Not digitally signed · Estimated work: 0.5 h

Will break when VBScript is disabled

Needs testing

Creates VBScript's regular-expression object

FormulaPrint, GETFORMULA, line 15

```
Set re = CreateObject("VBScript.RegExp") 'late binding
```

Why it matters. CreateObject("VBScript.RegExp") loads the pattern-matching engine that ships with VBScript. Microsoft says this keeps working on Microsoft 365 Version 2508 (build 19127.20154) or later, where RegExp is built into VBA, but fails on older Office builds once Windows disables VBScript (Microsoft expects that around 2026-27).

What to do. Check the Office build on every PC that uses this file (File > Account > About). On Microsoft 365 Version 2508 (build 19127.20154) or later no change is needed. Where any PC runs an older build or a perpetual Office (2019, 2021, 2024 LTSC), test with VBScript disabled, and plan the update to 2508+.

10. File 0018.xlsm Test

Excel macro-enabled workbook (.xlsm) · 6 modules, 56 lines of code · Not digitally signed · Estimated work: 0.5 h

Will break when VBScript is disabled

Needs testing

References "Microsoft VBScript Regular Expressions"

(library references)

Microsoft VBScript Regular Expressions 5.5 (C:\Windows\System32\vbscript.dll\3)

Why it matters. The project is linked to VBScript's regular-expression library. Microsoft says projects with this reference keep working on Microsoft 365 Version 2508 (build 19127.20154) or later, but will not run on older Office builds once Windows disables VBScript.

What to do. Confirm every user is on Microsoft 365 Version 2508 (build 19127.20154) or later. If not, test on a PC with VBScript disabled. On 2508+ the reference can be removed and the built-in RegExp used instead.

11. File 0004.xlsm Test

Excel macro-enabled workbook (.xlsm) · 20 modules, 679 lines of code · Not digitally signed · Estimated work: 2–6 h

Breaks on 64-bit Office

Needs testing

Uses Windows Common Controls (TreeView, ListView and similar)

(library references)

Microsoft Windows Common Controls 6.0 (SP6) (C:\Windows\system32\MSCOMCTL.OCX)

Why it matters. A 64-bit version of these controls (mscomctl.ocx) only ships with Microsoft 365 builds from July 2017 (Version 1707) onwards. Older 64-bit Office cannot load them, and the controls have had several breaking updates.

What to do. Test the forms on 64-bit Office at your current build; consider replacing the controls with standard UserForm controls.

12. File 0017.xlsm Test

Excel macro-enabled workbook (.xlsm) · 7 modules, 216 lines of code · Not digitally signed · Estimated work: 0.5–1 h

Breaks on 64-bit Office

Needs testing

Connects with the old Jet database provider

Module1, GetRecordSet, line 25

```
.Provider = "Microsoft.Jet.OLEDB.4.0"
```

Why it matters. The connection string names Microsoft.Jet.OLEDB.4.0, which exists only in 32-bit form. In 64-bit Office the connection fails with "Provider cannot be found".

What to do. Use Provider=Microsoft.ACE.OLEDB.12.0 (or 16.0), and make sure the Access Database Engine matching the Office bitness is installed.

13. File 0015.xlsm Test

Excel macro-enabled workbook (.xlsm) · 3 modules, 97 lines of code · Not digitally signed · Estimated work: 2–8 h

Unsupported, may stop working

Needs testing

Automates Internet Explorer

Module1, getNBAdata, line 23

```
Set ie = CreateObject("InternetExplorer.Application")
```

Why it matters. Internet Explorer 11 was retired on 15 June 2022 and is not offered as an app on Windows 11. Driving it from a macro may still work on some PCs today, but Microsoft no longer supports it, a Windows update can switch it off without notice, and many websites no longer work in it.

What to do. Replace the browser automation with direct web requests (MSXML2.XMLHTTP or WinHttp) or the site's API, or move it to a supported tool such as Power Automate, or Selenium with Edge.

14. File 0021.xlsm Review

Excel macro-enabled workbook (.xlsm) · 29 modules, 702 lines of code · Not digitally signed · Macro project locked for viewing (the code was still readable and was checked) · 5 links to other workbooks · Estimated work: 1.5–5 h

Security-relevant

Needs testing

Password or key written in the code 5 places

CreditInformation, ExtractReports, line 23

```
cnx.ConnectionString = "UID=sa;PWD=....;DRIVER={SQL Server};Server=" & Range("DB_SERVER").Value &
";Database=" & Range("DB_NAME").Value & ";"
```

CreditInformation, ExtractGlobalPortFolio, line 99

```
cnx.ConnectionString = "UID=sa;PWD=....;DRIVER={SQL Server};Server=" & Range("DB_SERVER").Value &
";Database=" & Range("DB_NAME").Value & ";"
```

CreditInformation, ExtractDelinquency, line 166

```
cnx.ConnectionString = "UID=sa;PWD=....;DRIVER={SQL Server};Server=" & Range("DB_SERVER").Value &
";Database=" & Range("DB_NAME").Value & ";"
```

CreditInformation, ExtractDelinquencyPerSectorReport, line 224

```
cnx.ConnectionString = "UID=sa;PWD=....;DRIVER={SQL Server};Server=" & Range("DB_SERVER").Value &
";Database=" & Range("DB_NAME").Value & ";"
```

CreditInformation, ExtractDelinquencyWithSectors, line 460

```
cnx.ConnectionString = "UID=sa;PWD=....;DRIVER={SQL Server};Server=" & Range("DB_SERVER").Value &
";Database=" & Range("DB_NAME").Value & ";"
```

Why it matters. A password, key or token is written into the macro as plain text. Anyone who can open the file can read it in the VBA editor (a locked project only slows them down), and it travels with every copy of the file. The value is masked in this report.

What to do. Find out whether it is still in use. If it is, change it, and have the macro ask for it or read it from a protected store (for example Windows Credential Manager) instead of the code.

15. File 0019.xlsm Review

Excel macro-enabled workbook (.xlsm) · 5 modules, 146 lines of code · Not digitally signed · Runs automatically: ResumeMatcher.Auto_Open · Estimated work: 0.5 h

Security-relevant

Runs automatically when the file opens (Auto_Open)

ResumeMatcher, Auto_Open, line 19

```
Public Sub Auto_Open()
```

Why it matters. This procedure runs on its own when the file is opened or closed, if macros are allowed, without anyone clicking anything. Security teams treat auto-run macros as the highest-risk kind.

What to do. Review what it does. If the file is trusted, sign the macros with your organisation's code-signing certificate so a "signed macros only" policy can allow it.

Security-relevant

Needs testing

Auto-run macro in a file that can start programs or reach the internet

ResumeMatcher, Auto_Open, line 19

```
Public Sub Auto_Open()
```

Why it matters. The file runs code automatically when opened, and elsewhere its code starts programs, talks to the internet or changes macro code. That combination is the pattern security tools look for, so review it first.

What to do. Have someone who knows the file confirm what the auto-run code calls, before the file is trusted or signed.

Security-relevant

Needs testing

Starts another program

ResumeMatcher, RunMatcher, line 103

```
Set execObj = wsh.Exec(cmd)
```

Why it matters. This line starts a program or opens a file with another program. Security teams review this, and it only works where that program exists. If your security baseline turns on the Microsoft Defender rule "Block all Office applications from creating child processes", this line is blocked.

What to do. Confirm the program is present on every PC and allowed by your application-control policy; if the Defender rule is on, agree an exclusion or move this step out of the macro.

16. File 0012.xlsm Review

Excel macro-enabled workbook (.xlsm) · 5 modules, 166 lines of code · Not digitally signed · Estimated work: 0.5–1.5 h

Security-relevant

Needs testing

Can send email without Outlook (CDO)

Module3, SendEmails_Gmail, line 32

```
Set objMail = CreateObject("CDO.Message")
```

Why it matters. The macro creates a CDO email message, which can send mail straight to a mail server without Outlook. Security teams want to know about documents that send mail. (CDO is sometimes used only to read or decode messages; check what this code does with it.)

What to do. Confirm the mail server and credentials it uses; many Microsoft 365 tenants now block basic-authentication SMTP.

Security-relevant

Needs testing

Password or key written in the code

Module3, SendEmails_Gmail, line 41

```
.Item("http://schemas.microsoft.com/cdo/configuration/sendpassword") = "...."
```

Why it matters. A password, key or token is written into the macro as plain text. Anyone who can open the file can read it in the VBA editor (a locked project only slows them down), and it travels with every copy of the file. The value is masked in this report.

What to do. Find out whether it is still in use. If it is, change it, and have the macro ask for it or read it from a protected store (for example Windows Credential Manager) instead of the code.

17. File 0023.xlsm Plan

Excel macro-enabled workbook (.xlsm) · 6 modules, 145 lines of code · Not digitally signed · Estimated work: 0.5–1 h

Security-relevant

Needs testing

Calls the Windows API (2 declarations: shell32, winspool.drv)

模块1, top of module, line 16

```
Declare PtrSafe Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" ( ByVal hwnd As Long, ByVal lpszOp As String, ByVal lpszFile As String, ByVal lpszParams As String, ByVal lpszDir As String, ByVal FsShowCmd As Long ) As Long
```

Why it matters. The macros call functions inside Windows directly (Declare statements). This is normal in older business macros, but it is also how malicious macros work. If your security baseline turns on the Microsoft Defender rule "Block Win32 API calls from Office macros", these calls are blocked.

What to do. Check whether your Defender attack surface reduction policy blocks Win32 API calls from Office macros. If it does, these macros need an exclusion or a rewrite.

Maintenance risk

Needs testing

Needs third-party software: Adobe Acrobat 10.0 Type Library

(library references)

```
Adobe Acrobat 10.0 Type Library (C:\Program Files (x86)\Adobe\Acrobat DC\Acrobat\acrobat.tlb)
```

Why it matters. The project is linked to a library that does not come with Windows or Office. On PCs without that software, the reference shows as MISSING and macros fail to compile.

What to do. Confirm the software is installed (in the same 32/64-bit flavour as Office) wherever this file is used.

Maintenance risk

Needs testing

Handle or pointer declared As Long (hwnd, the return value)

模块1, top of module, line 16

```
Declare PtrSafe Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" ( ByVal hwnd As Long, ByVal lpszOp As String, ByVal lpszFile As String, ByVal lpszParams As String, ByVal lpszDir As String, ByVal FsShowCmd As Long ) As Long
```

Why it matters. This Declare is marked PtrSafe, but handle or pointer arguments are declared As Long. In 64-bit Office handles and pointers are 64 bits wide, so this can give wrong results or crash Excel.

What to do. Declare handle and pointer arguments (and handle return values) As LongPtr.

18. File 0003.xlsm Plan

Excel macro-enabled workbook (.xlsm) · 3 modules, 44 lines of code · Not digitally signed

Maintenance risk

Needs testing

Drives Outlook

Module1, VBmail, line 37

```
Set emailApplication = CreateObject("Outlook.Application")
```

Why it matters. Controlling Outlook from a macro only works with classic Outlook for Windows. The new Outlook for Windows does not support VBA or COM automation, so this stops working for anyone moved to the new Outlook.

What to do. Keep users of this file on classic Outlook, or move the email step to Microsoft Graph or Power Automate before switching them.

19. File 0007.xlsm Plan

Excel macro-enabled workbook (.xlsm) · 2 modules, 32 lines of code · Not digitally signed · Estimated work: 0.5 h

Maintenance risk

Needs testing

Path in one person's user folder

Sheet1, ExtractDataToSingleFile, line 8

```
Set objectGetFolder = objectFileSys.GetFolder("C:\Users\[user]\OneDrive\Desktop\My_Projects\Open_Source_Contribution\py.ms365\Tutorial04-Excel-Macro-Extract-Data-from-Multiple-Excel-Files-4methods\src") ' The folder location for the s...
```

Why it matters. The code names a folder inside one person's Windows profile. It fails for every other user, and for that user on a new PC.

What to do. Build the path at run time (Environ("USERPROFILE"), ThisWorkbook.Path) or read it from a settings cell.

20. File 0008.xlsm Plan

Excel macro-enabled workbook (.xlsm) · 5 modules, 123 lines of code · Not digitally signed · Estimated work: 0.5 h

Maintenance risk

Needs testing

Path on a named network share

Módulo1, ImportarICMSST, line 21

```
caminhoPasta = "\\[server]\Users\ACESSO INTERNO\DOCUMENTOS FISCAIS\XML ENTRADA\"
```

Why it matters. The code names a server and share. It breaks when that server is renamed, retired or moved to SharePoint or OneDrive.

What to do. Put the path in a settings cell or named range so it can be changed without editing code, and list it in your file-share migration plan.

21. File 0020.xlsm Plan

Excel macro-enabled workbook (.xlsm) · 4 modules, 16 lines of code · Digitally signed (legacy; validity not checked) · Estimated work: 0.5 h

Maintenance risk

Needs testing

Needs third-party software: AcroBrokerLib

(library references)

```
AcroBrokerLib (C:\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\AcroBroker.exe)
```

Why it matters. The project is linked to a library that does not come with Windows or Office. On PCs without that software, the reference shows as MISSING and macros fail to compile.

What to do. Confirm the software is installed (in the same 32/64-bit flavour as Office) wherever this file is used.

22. **File 0026.xlsm** Plan

Excel macro-enabled workbook (.xlsm) · 3 modules, 59 lines of code · Not digitally signed · Estimated work: 0.5 h

Maintenance risk Needs testing **Needs other software installed: SAP GUI scripting**

VBA_Script, SAP_Main, line 7

```
Set SapGuiAuto = GetObject("SAPGUI")           'Get the SAP GUI Scripting object
```

Why it matters. The macro starts a program or component that does not come with Windows or Office, by name. On PCs without that software (or with a version for the other bitness, 32- or 64-bit), this line fails with "ActiveX component can't create object".

What to do. Confirm the software is installed and licensed on every PC that runs this file, in the same 32/64-bit flavour as Office, and add it to your PC build.

23. **File 0005.xlsm** No action

Excel macro-enabled workbook (.xlsm) · 6 modules, 169 lines of code · Not digitally signed · Runs automatically: ThisWorkbook.Workbook_Open

Security-relevant **Runs automatically when the file opens (Workbook_Open)**

ThisWorkbook, Workbook_Open, line 1

```
Private Sub Workbook_Open()
```

Why it matters. This procedure runs on its own when the file is opened or closed, if macros are allowed, without anyone clicking anything. Security teams treat auto-run macros as the highest-risk kind.

What to do. Review what it does. If the file is trusted, sign the macros with your organisation's code-signing certificate so a "signed macros only" policy can allow it.

Files with macros and nothing found

Nothing in these files' macros matched a known breaking change or risk.

File	Format	Lines of code	Signed
File 0001.xlsm	Excel macro-enabled workbook (.xlsm)	259	No
File 0002.xlsm	Excel macro-enabled workbook (.xlsm)	207	No

Files without macros, or not checked

File	Result	Note
File 0006.xlsm	No macros	No VBA macros in this file.

How this was checked, and its limits

Each file was opened read-only. Its VBA macro project was extracted and every module was parsed as code, so a word in a comment, or in a message to the user, is never counted as a dependency. A string counts only where the code uses it for what it names: an object passed to CreateObject, a command passed to Shell, a file path, a password. Code is read the way 64-bit Office on Windows compiles it: the old-Office branch of `#If VBA7`, and anything under `#If Mac`, is left out.

- **Read, not run.** Nothing was executed. Code that builds an object name or a command at run time, or loads code from elsewhere, can only be partly analysed; where it matters it is reported as "needs testing". Whether the macros do the right thing for your business was not checked.
- **Only these files.** Macros kept elsewhere (each user's Personal.xlsm, add-ins installed on PCs, Word's Normal.dotm) were not checked unless they were among the files sent.
- **Locked projects.** A password-protected project whose code cannot be read is reported as not checked. (Many locked projects still keep readable code, and those were checked.)
- **Compiled-only code.** A module that keeps compiled code but no readable source is reported, not analysed.
- **Passwords and keys.** Values that look like passwords or keys are masked (••••) in every output. Whether one is still in use was not checked.
- **Signatures.** The report says whether a signature is present. Whether it is valid, and whether your PCs trust the certificate, was not checked.
- **Internet mark.** The "downloaded from the internet" mark belongs to a copy of a file. The copies checked here may not carry it, but copies sent by email or downloaded from the web will, and Office blocks their macros.
- **Your settings decide some outcomes.** Office build and channel, 32- or 64-bit Office, VBScript on or off, Trust Center and Defender policies all change what breaks. Findings say which one each depends on.
- **Estimates.** Hours are planning ranges from typical times per kind of change, not a quote or a promise.
- **Formats read:** Excel .xlsm, .xltm, .xlsb, .xlam, .xls, .xlt, .xla; Word .docm and .dotm; PowerPoint .pptm, .potm, .ppsm and .ppam. Legacy Word .doc and PowerPoint .ppt are not read.