


Excel(lent) Malicious Malware Analysis

Evan H. Dygert, SANS Certified Instructor

Dygert Consulting, Inc.



About Me

- GSE #43
- Software Developer
- Digital Forensicator
- Malware Analyst
- SANS CyberGuardian Red and Blue teams

Goals

- Show how Excel 4 macros work and their capabilities.
- Demonstrate how to analyze them.
- Provide samples for you to practice on (learn by doing).

Excel 4 Macros

- Ancient technology (about 30 years).
- Still not reliably detected by AV.
- Can call Windows API.
- Can run VBA functions.
- Can run arbitrary programs.
- Excel 4 macros use macrosheets.
- Also called XLM macros.
- Still work in latest versions of Excel.

Excel 4 Macro Example

- Create blank workbook.
- Create macrosheet (Ctrl-F11).
- Write macros using syntax similar to formulas.
- Excel 4 Macro functions reference:
<https://www.myonlinetraininghub.com/excel-4-macro-functions>

Excel Settings

- File->Options ->Trust Center->Trust Center Settings
 - Trusted Publishers/Locations/Documents/Add-in Catalogs
 - Macro Settings
 - File Block Settings
- Show Developer menu option.
File->Options ->Customize Ribbon->Check "Developer" tab.

Linux VM

- REMnux by Lenny Zeltser
<https://remnux.org/>
- olevba and XLMMacroDeobfuscator (aka xlmdeobfuscator) are already installed in REMnux.
- Make sure networking is set to Host Only mode after installing all your tools.
- Take a snapshot of the VM.
- Make a note of the IP address of the VM.

Windows VM

- Flare VM created by FireEye
<https://github.com/fireeye/flare-vm>
- Excel
- Make sure networking is set to Host Only mode after installing all your tools.
- Set Default Gateway and DNS server to the IP address of the Linux VM and hardcode the Windows VM IP address to something in the same subnet.
- Take a snapshot of the VM.

Sample 1: Triage

- olevba says suspicious.
- xlmdeobfuscator output also suspicious but behavior not clear.

Sample 1 Analysis: Excel

1. Can't go to Auto_Open cell.
2. Unhide hidden sheets.
3. View Auto_Open cell.
4. Show contents of apparently empty cells by changing text colors.
5. Scroll down to see first instruction (H23).
6. Macro makes a copy of the workbook.

Sample 1 Analysis: Excel (cont.)

7. Copy is named `..\Nioka.meposv`
8. Jumps to `Nolaert!AK161`
9. Go to cell `Nolaert!AK161`. Change colors for this sheet and scroll down to `AK703` to see first instruction.
10. Right-click->Run ->Step into EXEC (`AK703`)
11. Won't run because macros not enabled.
12. Add `PAUSE()` at `AK702` above EXEC and click "Enable Content". Macro will be paused on this line.

Sample 1 Analysis: Excel (cont.)

13. Right-click->Run ->Step into Exec.
14. Click Evaluate to see what is being executed (tar????!!!).
15. Step over until 2nd EXEC (AK711) and Evaluate until EXEC arguments are visible.
16. Note name of file(..\xl\media\image2.bmp), then close Excel.
17. Unzip the .xlsm file and examine the image2.bmp file using a tool like PEStudio to see that it is a DLL.

Sample 1: Summary

- Hidden worksheets and macrosheets are common.
- Unhide them to view the code and data.
- Text colors may be used to hide cell contents.
- XLM macros can be debugged in Excel.
- Use `PAUSE()` to stop execution then right-click->Run->Step into->Continue on next line to continue execution to the next `PAUSE`.

Sample 2: Triage

- olevba says suspicious.
- xlmdeobfuscator output also suspicious.
- Both tools show a lot of obfuscation using FORMULA() and CHAR() functions referencing the W86 cell.

Sample 2 Analysis: xlmdeobfuscator

- xlmdeobfuscator -n output
 - W86 should equal 247.12 to deobfuscate the rest of the script.
 - Calls to FORMULA create the second stage script at B134.
 - GET.WORKSPACE used for anti-sandbox/anti-debugging (see XLM Macro Reference).
 - GOTO(B134) at E110 runs deobfuscated script.

Sample 2 Analysis: Excel

1. Change text colors to view script.
2. Uses R1C1 reference style, change to A1 for convenience (File->Option->Formulas. Uncheck R1C1 reference style).
3. Put 247.12 in W86 and delete A81 to keep script from overwriting W86.
4. Follow the GOTOs until stage 2 is deobfuscated (first non-FORMULA/GOTO cell).

Sample 2 Analysis: Excel (cont.)

5. Replace cell E109 with PAUSE() to prevent hiding the active window and pause the macro.
6. Enable Content. Macro will pause at E109. (May need to restart Excel to show the button).
7. Analyze deobfuscated script at B134.

Sample 2: Summary

- Switching formula notation may be helpful.
- FORMULA and CHAR used heavily for obfuscation.
- Text colors used to hide macro text.
- GET.WORKSPACE used for anti-sandbox/anti-debugging.
- Use xlmdeobfuscator to speed up and simplify analysis.
- Put PAUSE() at key points.
- Let the script deobfuscate itself.

Sample 3: Triage

- olevba says suspicious. Hidden Auto_Open name. Also shows what appears to be a non-printable name.
- xlmdeobfuscator -x shows the part of the script that does deobfuscation but -n fails. Also reports that start cell is AU3287.

Sample 3 Analysis: Excel

1. Formulas->Name Manager shows no names because they are hidden.
2. Add this script to ThisWorkbook and run it to show names:

```
Sub UnhideAllNames()  
    Dim n As Name  
  
    For Each n In ActiveWorkbook.Names  
        n.Visible = True  
    Next n  
End Sub
```

Sample 3 Analysis: Excel (cont.)

3. Open Formulas->Name Manager and change the seemingly blank name to "aa". (I usually use two letter names for simplicity).
4. Save As a new file to avoid modifying the original file.
5. Go to AU3287 (aka Auto_Open/aa) and examine that line. Macro will halt if anti-sandbox/anti-debugger check fails. Otherwise, SUM(3,4) will be the result.
6. Change the value of AU3287 to 7 to defeat defenses and save again.
7. Examine code to find possible calls to deobfuscation routine(s)...

Sample 3 Analysis: Finding Deobfuscation Routine(s)

1. Look for repeated calls to the same routine.
2. Save output of `xlmdeobfuscator -x` to `macro.txt`.
3. Filter out irrelevant macro lines.

```
egrep -v 'SET|WHILE|NEXT|RETURN|MIN|HALT' macro.txt
```

4. Only one candidate! (OPmOHyaBMDZM).

Sample 3 Analysis: Finding Deobfuscation Routine(s) (cont.)

5. The instructions immediately before and after the candidate deobfuscation routine are further evidence that we have found the real routine.
6. In this case, the address of the cell containing the decoded script is immediately before the call to the deobfuscator and the call to the deobfuscated script is immediately after (3 times).
7. The three deobfuscated scripts are named DNGGrxPtt, BpMhzoLtvh, and EHBZcDhoF.
8. This is a common pattern for deobfuscators.

Sample 3 Analysis: Deobfuscation

1. Now that a possible deobfuscation routine has been found, add `PAUSE()` after each call to it.
2. Press Enable Content to allow the macro to execute to the first `PAUSE()` immediately before the call to `DNGGrxPtt`.
3. Use the name dropdown to jump to `DNGGrxPtt` (HF21404) and analyze the first deobfuscated script.
4. Note the use of `GET.WINDOW/GET.WORKSPACE` to hinder analysis.

Sample 3 Analysis: Deobfuscation (cont.)

5. Delete lines that jump to HF21403 which closes the active window. DO NOT delete all anti-analysis lines.
6. Last line of 1st script jumps to AU3324 or AU3349 depending on platform. We want to analyze both cases. PAUSE() function already in place for both of those code chunks.
7. Replace the 2nd to last line of the 1st script (HF21433) with PAUSE().
8. Right-click Run on HF21404, Step into, then Continue to execute to the next pause before the jump to either AU3324 or AU3349.

Sample 3 Analysis: Deobfuscation (cont.)

9. To view the first branch, go to AU3324, right mouse click on that cell, step into it, and click Continue. Execution will be paused at the Pause statement we added earlier.
10. Another script (BpMhzoLtvh) has been decoded. Use name dropdown to view and analyze it.
11. To see what the other branch looks like, go to AU3349 and follow the same steps as for AU3324.
12. The 3rd script is now decoded (EHBZcDhoF). Use name dropdown to view and analyze it.

Sample 3: Summary

- May need to run VBA script to unhide names and fix them.
- Don't just delete anti-sandbox/anti-debugger lines. May need to replace with value for the sample to work.
- Add PAUSE() at strategic points for debugging.
- Use the names in the name dropdown to simplify navigating the script.
- Let the script deobfuscate itself.
- Rerun triage/analysis tools after fixing names/formulas.
- Collect all the IOCs! (user agents, IPs, URLs, etc.)

Recommendations

- Be wary of XLM macros.
- Find and use tools that can detect possibly malicious documents.
- Use Group Policy to set Trust Settings for MS Office.
<https://4sysops.com/archives/restricting-or-blocking-office-2016-2019-macros-with-group-policy/>
- If possible, block Excel 4 macros completely.
- Get comfortable with Excel debugging tools.

Conclusions

- You now have the tools/techniques to analyze XLM macros.
- Be flexible. Attackers are constantly changing obfuscation techniques.
- Malicious documents you find in your environment are high quality indicators for threat hunting and cyber threat intelligence.

Macro to Unhide All Sheets

```
Sub UnhideSheets()  
    Dim s As Worksheet  
  
    For Each s In Sheets  
        s.Visible = xlSheetVisible  
    Next  
End Sub
```

Note that the loop uses "Sheets" not "Worksheets" to also unhide macrosheets.

Macro to Unhide All Names

```
Sub UnhideAllNames()  
    Dim n As Name  
  
    For Each n In ActiveWorkbook.Names  
        n.Visible = True  
    Next n  
End Sub
```

References

- Writeups
 - <https://inquest.net/blog/2020/03/18/Getting-Sneakier-Hidden-Sheets-Data-Connections-and-XLM-Macros>
 - <https://isc.sans.edu/diary/25868>
 - <https://www.goggleheadedhacker.com/blog/post/21>
 - <https://inquest.net/blog/2019/01/29/Carving-Sneaky-XLM-Files>
 - <https://inquest.net/blog/2020/05/06/ZLoader-4.0-Macro-sheets->
 - <https://malware.pizza/tag/excel-4-0-macros/>
 - <https://malware.pizza/2020/05/12/evading-av-with-excel-macros-and-biff8-xls/>
- Excel 4 Macros Trend:
<https://www.sneakymonkey.net/2020/06/22/excel-4-0-macros-so-hot-right-now/>

References (cont.)

- EvilClippy: <https://github.com/outflanknl/EvilClippy>
- XLM Macro Reference: <https://www.myonlinetraininghub.com/excel-4-macro-functions>
- MS XLM Macro Reference: <https://support.microsoft.com/en-us/office/excel-functions-alphabetical-b3944572-255d-4efb-bb96-c6d90033e188>
- VBAWarning setting: <https://social.technet.microsoft.com/Forums/ie/en-US/f091b658-4a71-4f12-bd2d-aea9ac53a65d/enable-ms-excel-macrodisable-excel-macro-warning?forum=Office2016ITPro>
- <https://www.lastline.com/labsblog/evolution-of-excel-4-0-macro-weaponization/>

References (cont.)

- Macrome Excel document reader/writer:
<https://github.com/michaelweber/Macrome>
- Unhiding names:
<https://excelhelphq.com/how-to-unhide-or-delete-name-ranges-in-excel/>
- oletools (includes olevba)
<https://github.com/decalage2/oletools/wiki/Install>
- xlmdeobfuscator
<https://github.com/DissectMalware/XLMMacroDeobfuscator>
- <https://www.ablebits.com/office-addins-blog/2017/12/20/very-hidden-sheets-excel/>