Excel 4.0 Macro (XLM) Analysis

written by Mert SARICA | 1 June 2021

The DDE-based phishing attacks that started in 2017 have been replaced by Microsoft Excel 4.0 Macro (XLM) phishing attacks as of 2020. A little research would show that XLM macros have been introduced to the world since 1992, with the release of Microsoft Excel 4.0. VBA macros, which are frequently misused by threat actors, were first introduced with Excel 5.0 and are still supported in the latest version of Microsoft Office.

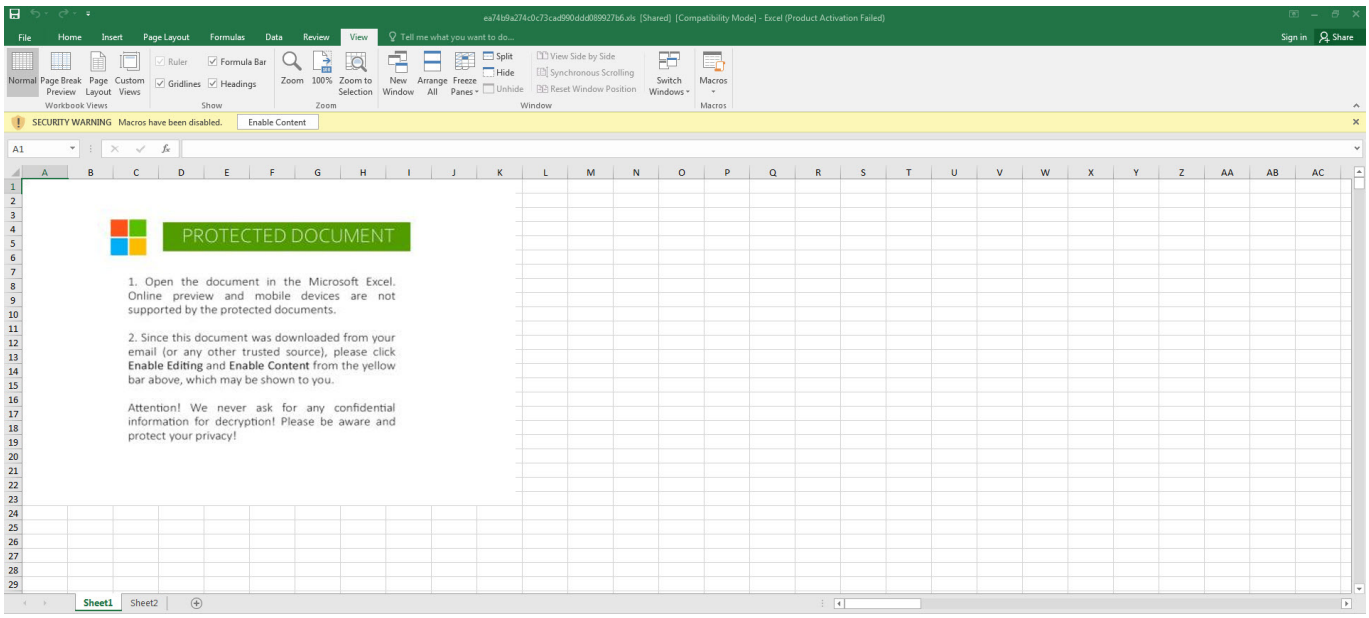
If my memory serves me correctly, the first technical article I read about XLM macros was this blog post from Outflank. As research on XLM macros started to reveal their existence, it began to attract attention not only from offensive security experts but also from threat actors. Soon enough, organizations began to experience phishing attacks that contained XLM macros. Due to the difficulty of detecting and analyzing XLM macros as compared to VBA macros, it is not an easy task.

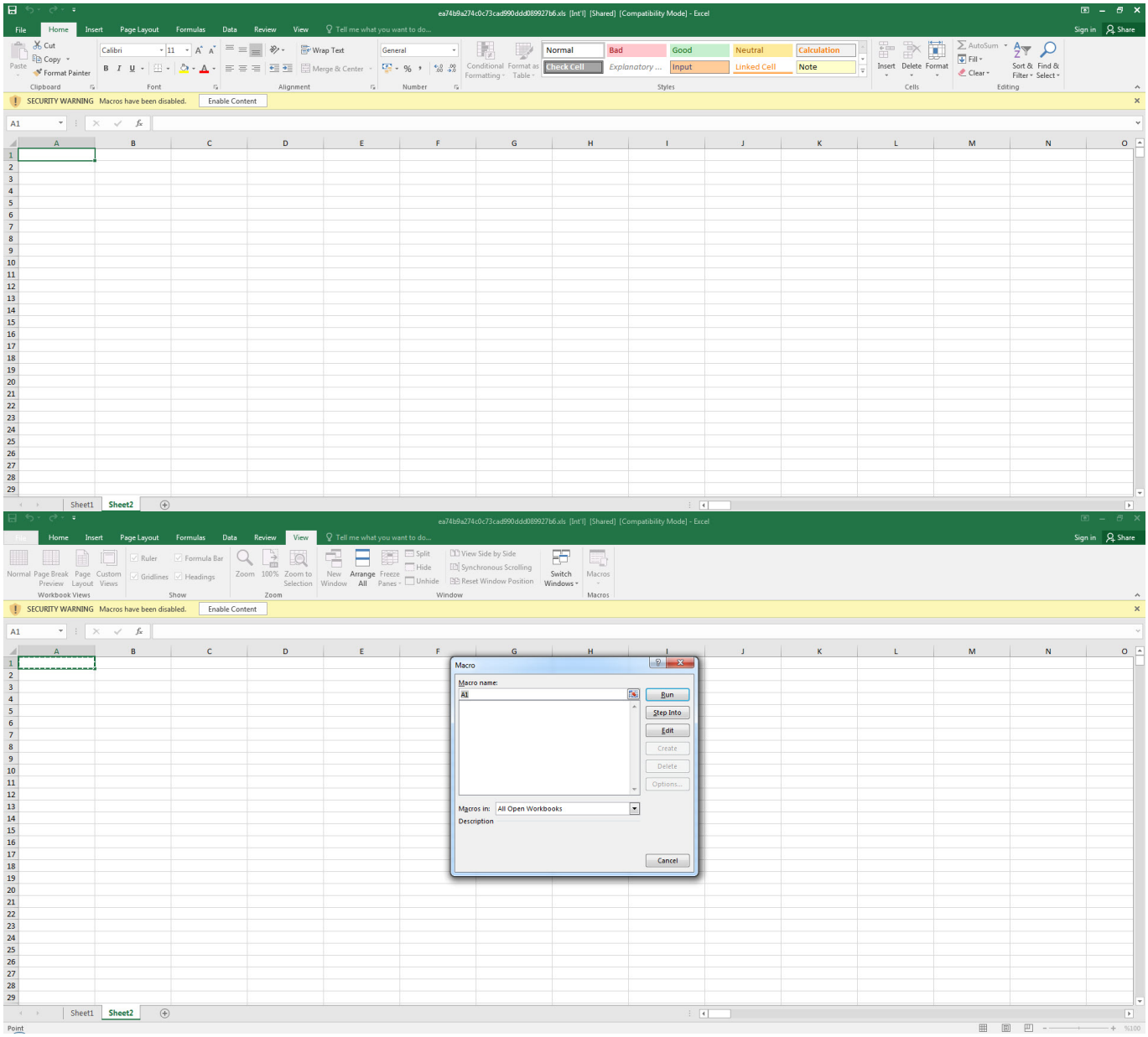
The difficulty in analyzing an Office file containing XLM macros, as I stated in my blog post titled "Microsoft Office Macro Analysis", is caused by the fact that they cannot be easily viewed from the Microsoft Office interface. As a result, the possibility of malicious XLM macro-containing Office files going unnoticed by inexperienced cybersecurity professionals (such as "This Office file is corrupted" or "Does not contain macros") increases. In order to show cybersecurity analysts how XLM macro-containing Microsoft Office files can be analyzed and to raise awareness about XLM macro-containing Microsoft Office files, I decided to write a blog post based on a real-life incident.

In May 2020, alarms began to be generated for many SMTP IP addresses from which hundreds of emails with sender addresses ending in @wp.pl were sent and blocked by security systems. Upon inspection, the emails had attachments with XLS extension, Excel files that have been randomly named. In such cases, one of the most important steps for cybersecurity analysts to take is to identify the addresses of the command and control centers in the malicious document, search for them in web traffic records, and block access throughout the organization.

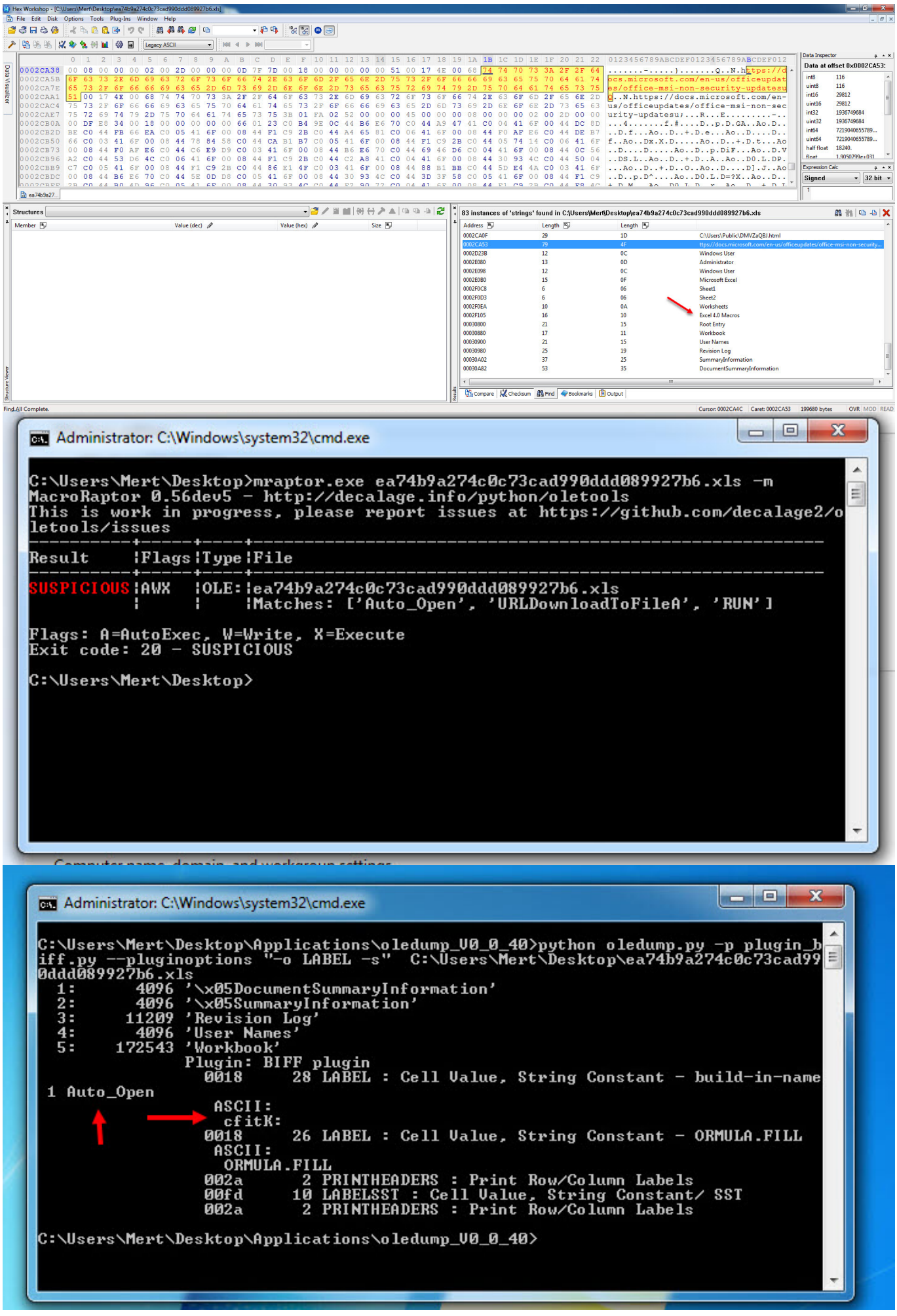
Of course, when the issue at hand is an Office file containing XLM macros, it’s possible that static and dynamic analysis performed by sandbox systems may be insufficient in the face of anti-sandbox techniques (e.g. Sandbox Detection). If the malicious Excel file in question is designed to detect when it is running in a sandbox, then the address of the command and control center will not be revealed during these analyses (VirusTotal, Hybrid-Analysis). In this case, the cybersecurity analyst’s job should be to take the malicious Excel file, copy it to a virtual system created for the purpose of malware analysis, and analyze it there.

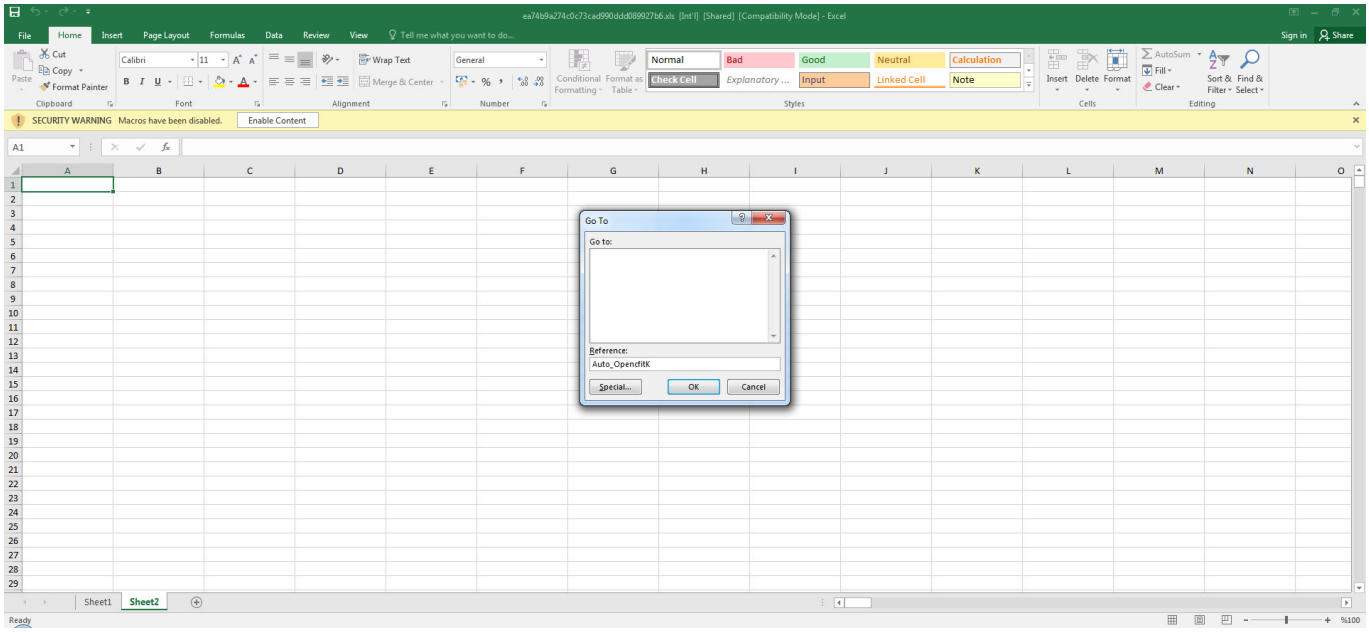
When running the Excel file in a virtual system, we are presented with two pages (Sheet1 and Sheet2). The first page contains a fake image/message indicating that we need to activate the macro to achieve its malicious intent, while the second page shows empty cells (which are not actually empty). Although Excel may warn us that there is a macro in the file, when we view the macro, it appears to be empty.
This kind of macro-based attack often called “Fileless attack” or “Living-off-the-land attack” because it doesn’t involve in injecting code into the system or download any malicious file. Instead, it makes use of the system’s legitimate tools in order to perform its malicious action, thus it’s harder to detect.



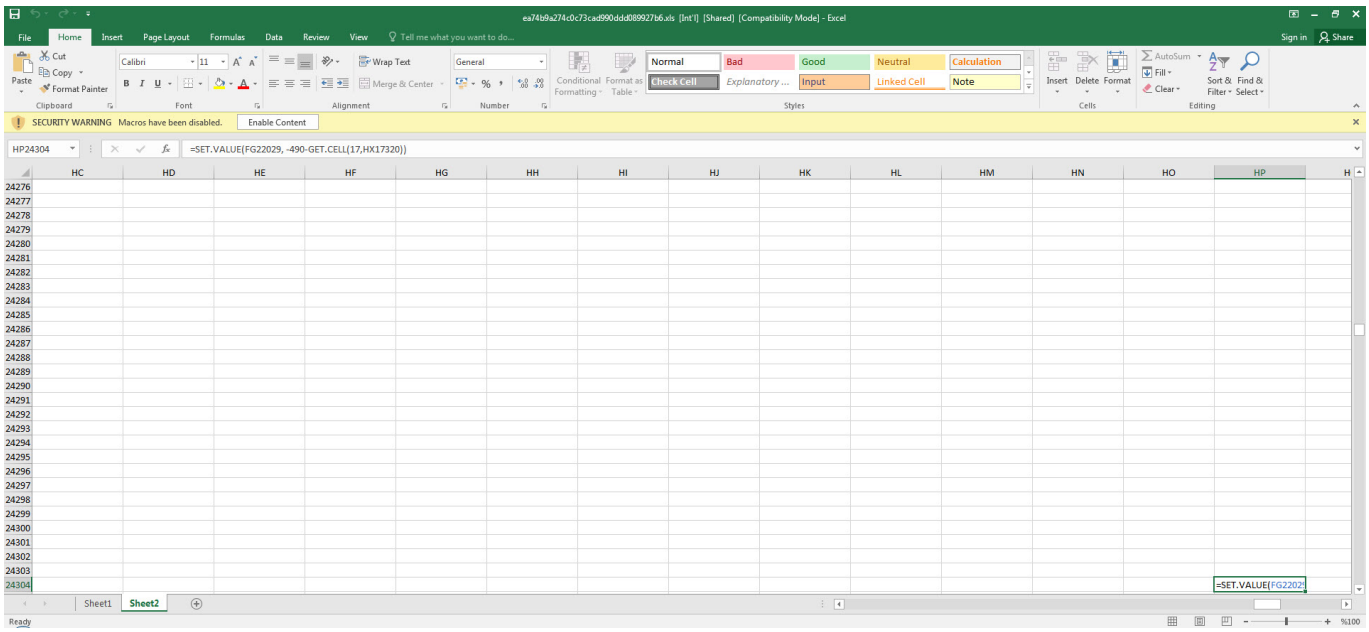


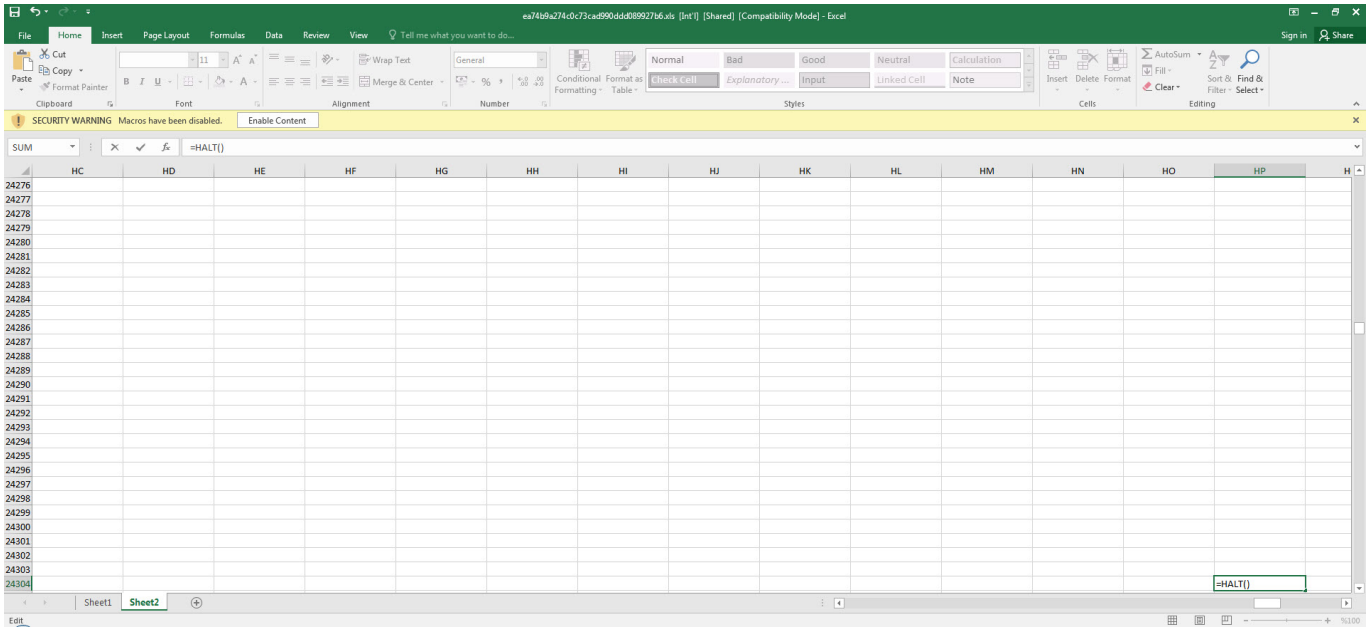
When opening the file with any hex editor and looking at the character strings contained within, we can see that, as suspected from the Excel 4.0 Macros series, it contains XLM macros. To be sure that the file contains a macro, when we analyzed the file with the mraptor tool, we could see that the file had a cell named Auto_Open, which was capable of running automatically, similar to the AutoOpen() function in a VBA macro. To learn the name of the cell and view and analyze it, I used Didier STEVENS‘ oledump tool. And by using (oledump.py -p plugin_biff.py –pluginoptions “-o LABEL -s” C:\Users\Mert\Desktop\ea74b9a274c0c73cad990ddd089927b6.xls) I found that the cell that was first run was named Auto_OpencfitK. This is a clever technique used by the malware developer to evade detection. Knowing that an analyst would use Go To (CTRL-G) in Excel to go to the cell named Auto_Open to start the analysis, the developer changed the name of the cell to Auto_OpencfitK


Hex Workshop - [C:\Users\Mert\Desktop\ea74b9a274c0c73cad990ddd089927b6.xls]
Structures
83 instances of 'strings' found in C:\Users\Mert\Desktop\ea74b9a274c0c73cad990ddd089927b6.xls
Administrator: C:\Windows\system32\cmd.exe
C:\Users\Mert\Desktop>mraptor.exe ea74b9a274c0c73cad990ddd089927b6.xls -m
MacroRaptor 0.56dev5 - http://decalage.info/python/oletools
This is work in progress, please report issues at https://github.com/decalage2/oletools/issues
Result | Flags | Type | File
SUSPICIOUS | AWX | OLE: | ea74b9a274c0c73cad990ddd089927b6.xls
Matches: ['Auto_Open', 'URLDownloadToFileA', 'RUN']
Flags: A=AutoExec, W=Write, X=Execute
Exit code: 20 - SUSPICIOUS
C:\Users\Mert\Desktop>
Administrator: C:\Windows\system32\cmd.exe
C:\Users\Mert\Desktop\Applications\oledump_V0_0_40>python oledump.py -p plugin_biff.py --pluginoptions "-o LABEL -s" C:\Users\Mert\Desktop\ea74b9a274c0c73cad990ddd089927b6.xls
1: 4096 '\x05DocumentSummaryInformation'
2: 4096 '\x05SummaryInformation'
3: 11209 'Revision Log'
4: 4096 'User Names'
5: 172543 'Workbook'
Plugin: BIFF plugin
0018 28 LABEL : Cell Value, String Constant - build-in-name
1 Auto_Open
ASCII:
cfitK:
0018 26 LABEL : Cell Value, String Constant - ORMULA.FILL
ASCII:
ORMULA.FILL
002a 2 PRINTHEADERS : Print Row/Column Labels
00fd 10 LABELSST : Cell Value, String Constant/ SST
002a 2 PRINTHEADERS : Print Row/Column Labels
C:\Users\Mert\Desktop\Applications\oledump_V0_0_40>



After we found out that the initial cell was an obfuscated macro consisting of 42 FORMULA statements and CHAR functions throughout the file, analyzing and solving each one of them one by one would have taken a significant amount of time. So I decided to proceed with debugging. By going to the Auto_OpencfitK cell and pressing ALT + F8, I then pressed the Step Into button, and Excel prompted me to allow the macro to run and then close and re-open the file. As soon as the file was opened, Excel quickly moved to the Auto_OpencfitK cell. To avoid missing this step, I changed the formula =SET.VALUE(FG22029, -490-GET.CELL(17,HX17320)) in that cell to =HALT() and this caused the macro to end. After this, I changed the =HALT() formula to =SET.VALUE(FG22029, -490-GET.CELL(17,HX17320)) and then by pressing the ALT + F8 on the cell, I was able to dynamically analyze the macro from the initial cell without any issues.

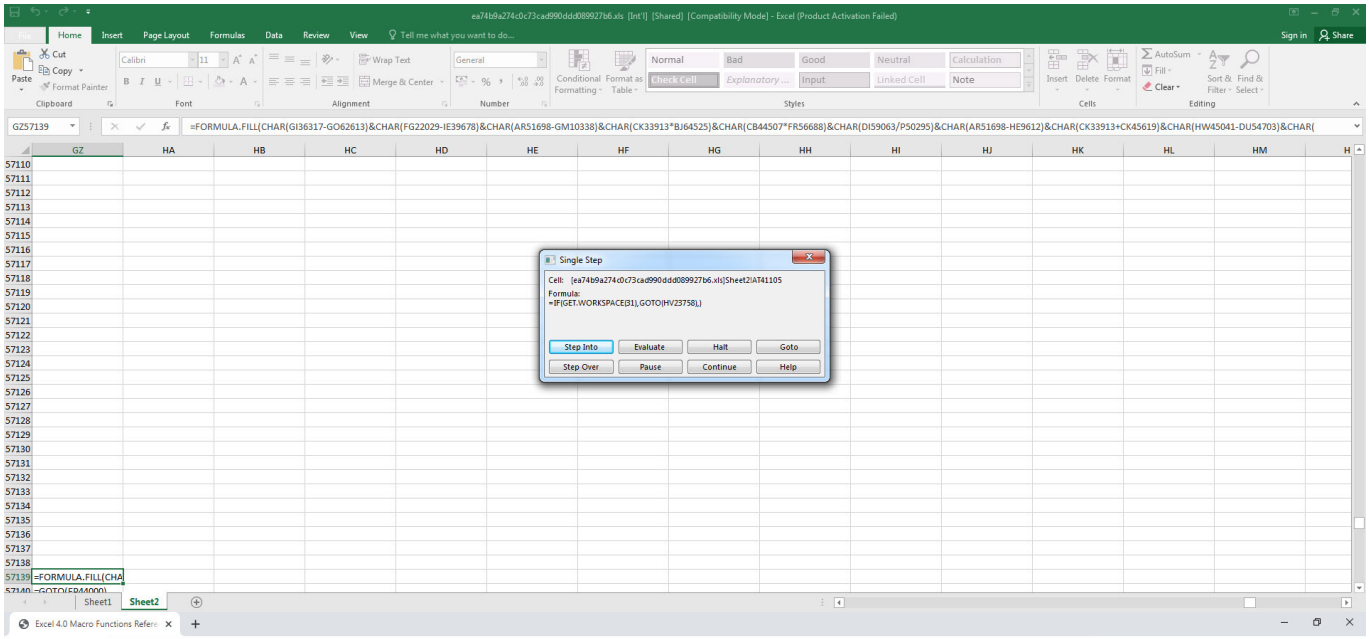


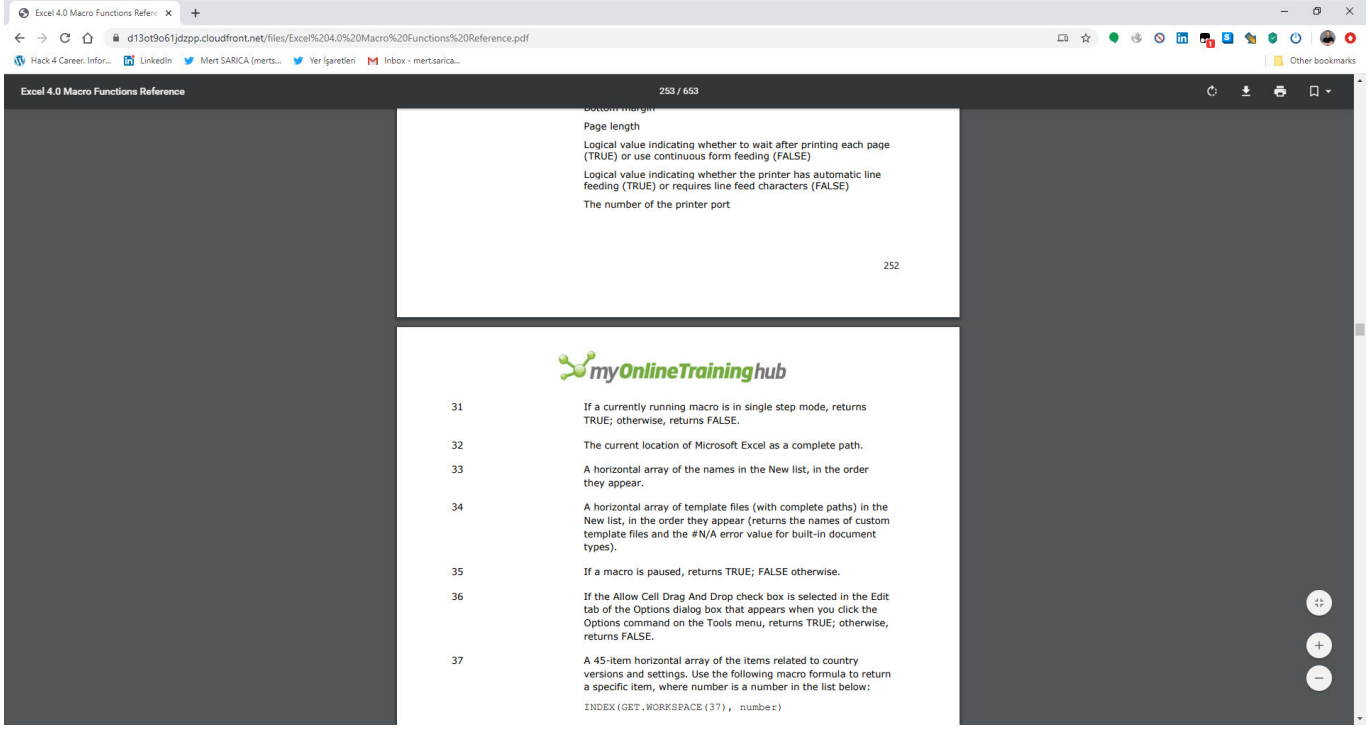


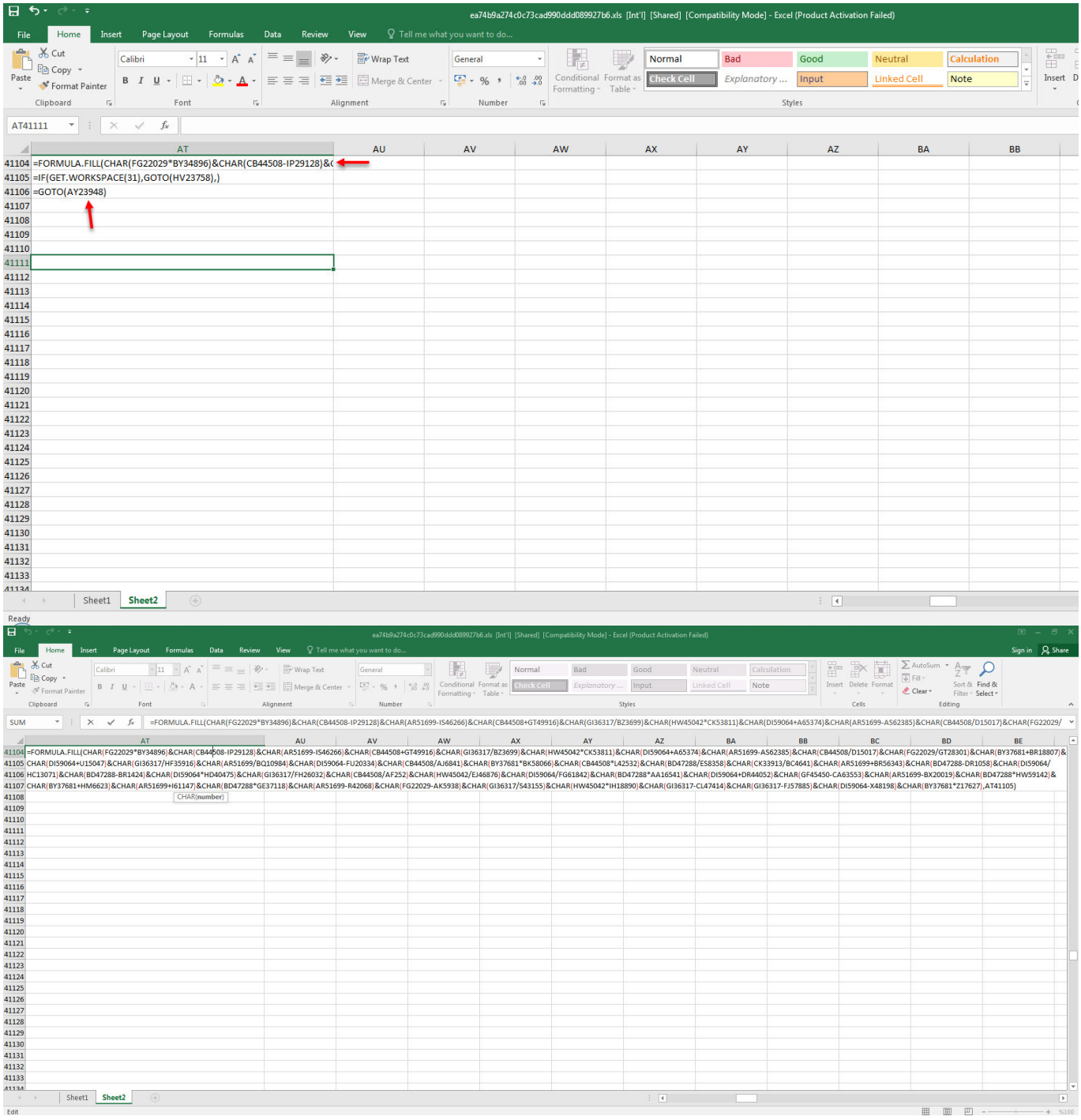
As I continued analyzing by using Step Into and Evaluate buttons, and decoding the hidden cells, I saw that the macro uses various controls against debugging and sandbox environments by using Excel 4.0 Macro Functions Reference document. When I reached the AT41104 cell, which is performing the debugging control, to bypass this control, I copied the =GOTO(AY23948) value in the next cell where it would continue if debugging is not detected.

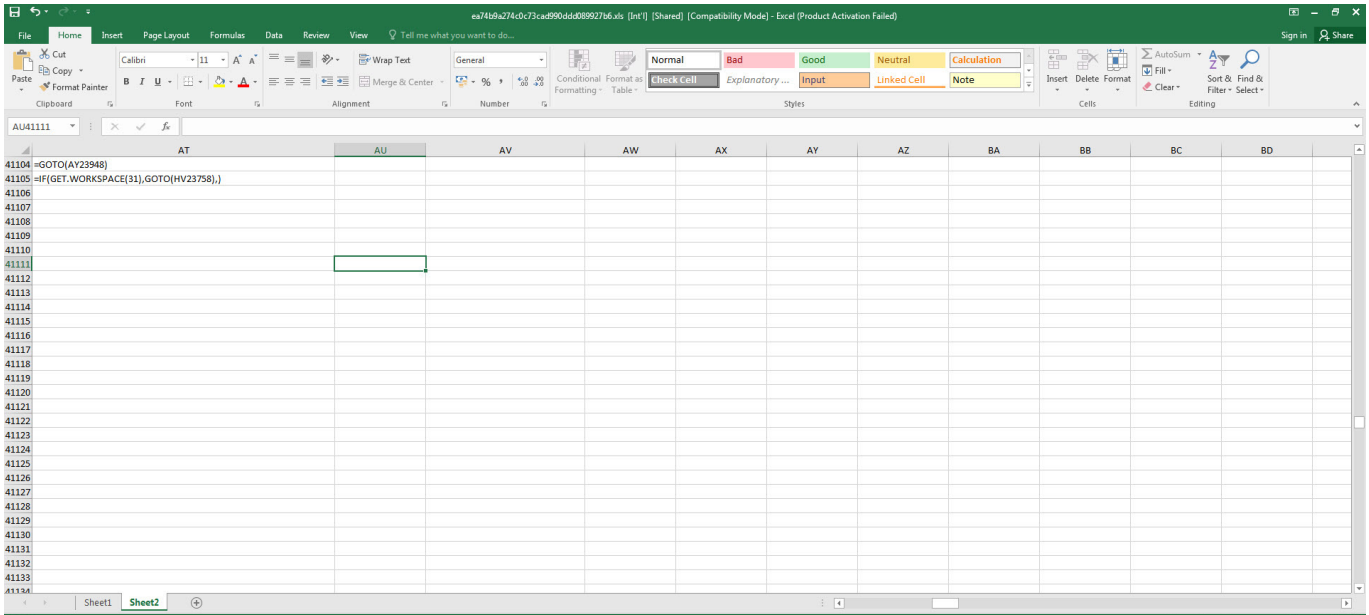
=IF(GET.WORKSPACE(31),GOTO(HV23758),) Is the macro in debugging mode? (Anti-debugging)
=IF(GET.WORKSPACE(19),,GOTO(HV23758),) Is a mouse present on the system? (Anti-sandbox)
=IF(GET.WORKSPACE(42),,GOTO(HV23758),) Can the system play sound? (Anti-sandbox)

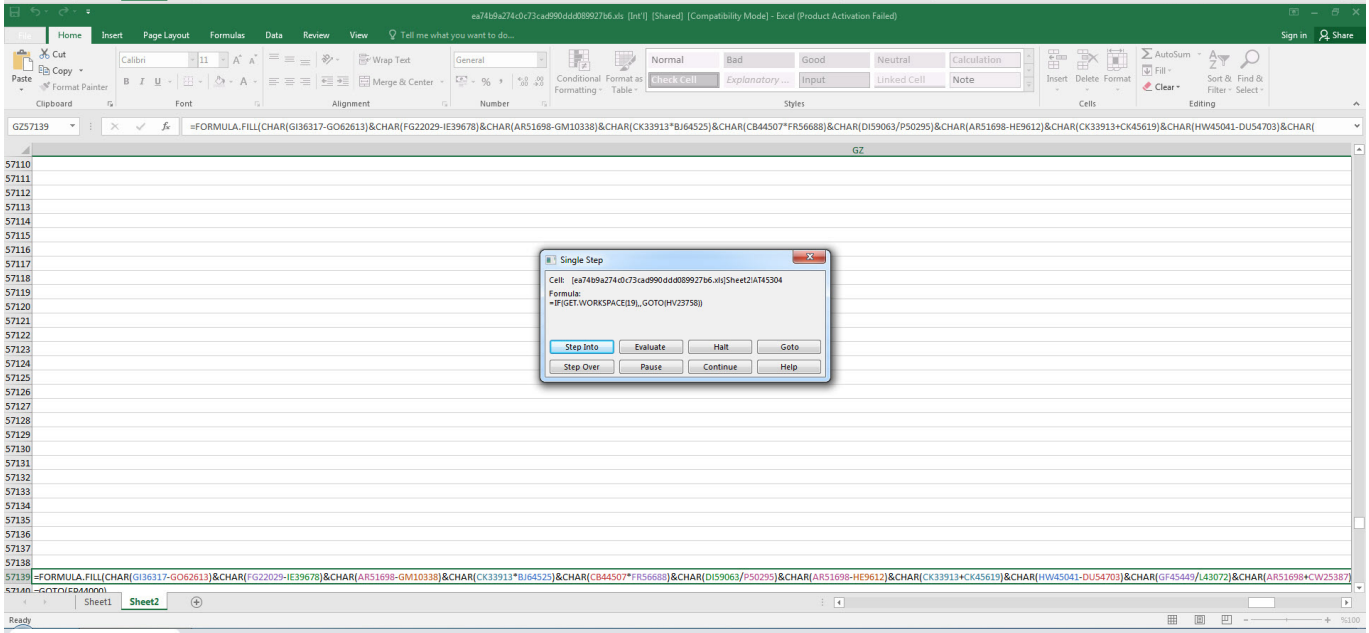
These statements or functions checks whether macro is running in a sandbox environment or on a real machine or if it’s in debugging mode.It also try to detect other anti-sandbox evasions like Mouse or sound. These checks used by malware developer to avoid detection, and prevent the macro from running when it’s running in an environment that the attacker doesn’t want it to run in.

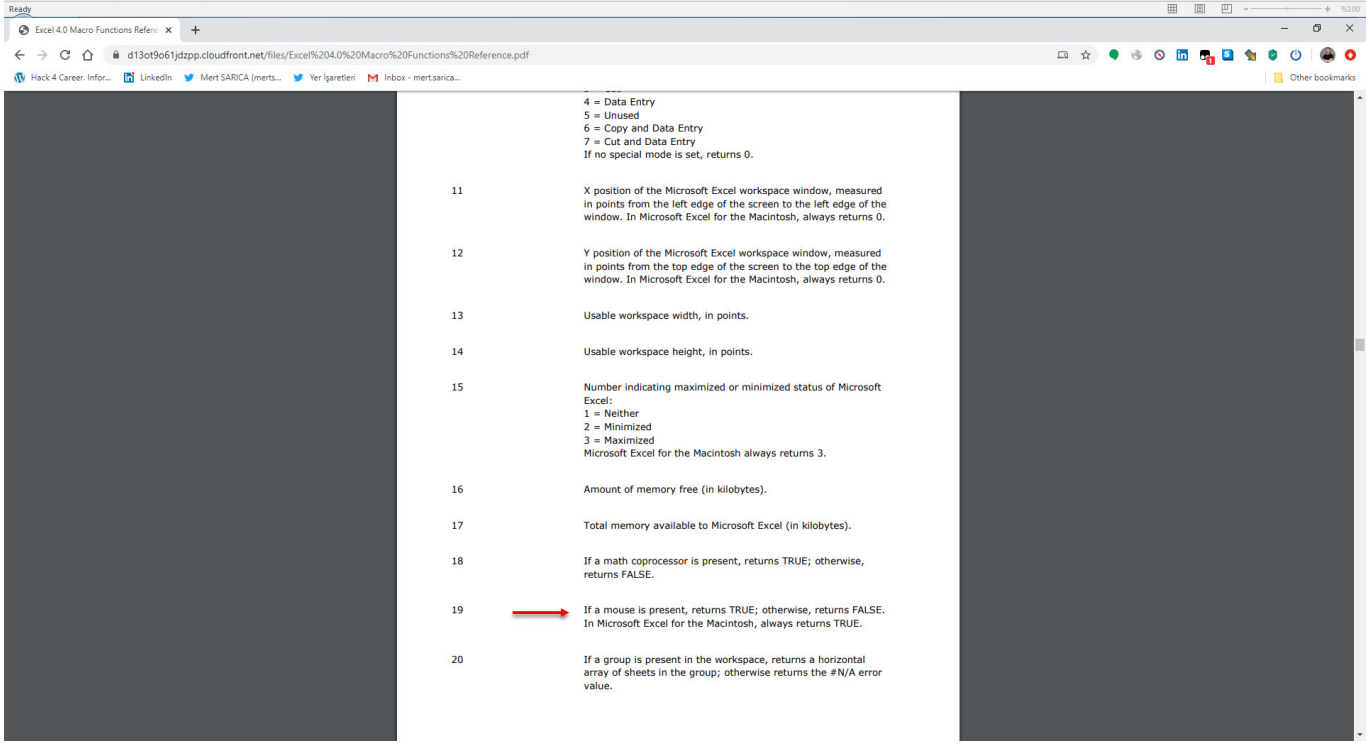

ea74b9a274c0c73cad990ddd089927b6.xls [Int'l] [Shared] [Compatibility Mode] - Excel (Product Activation Failed)
Home
Insert
Page Layout
Formulas
Data
Review
View
Tell me what you want to do...
Sign in
Share
Paste
Cut
Copy
Format Painter
Clipboard
Calibri
11
Font
Wrap Text
Merge & Center
Alignment
General
Number
Conditional Formatting
Format as Table
Normal
Bad
Good
Neutral
Calculation
Check Cell
Explanatory ...
Input
Linked Cell
Note
Styles
Insert
Delete
Format
Cells
AutoSum
Fill
Clear
Sort & Filter
Find & Select
Editing
GZ57139
=FORMULA.FILL(CHAR(GI36317-GO62613)&CHAR(FG22029-IE39678)&CHAR(AR51698-GM10338)&CHAR(CK33913*BJ64525)&CHAR(CB44507*FR56688)&CHAR(DI59063/P50295)&CHAR(AR51698-HE9612)&CHAR(CK33913+CK45619)&CHAR(HW45041-DU54703)&CHAR(
GZ
HA
HB
HC
HD
HE
HF
HG
HH
HI
HJ
HK
HL
HM
57139 =FORMULA.FILL(CHA
57140 =GOTO(ERA4000)
Sheet1
Sheet2
Single Step
Cell: [ea74b9a274c0c73cad990ddd089927b6.xls]Sheet2!AT41105
Formula:
=IF(GET.WORKSPACE(31),GOTO(HV23758),)
Step Into
Evaluate
Halt
Goto
Step Over
Pause
Continue
Help

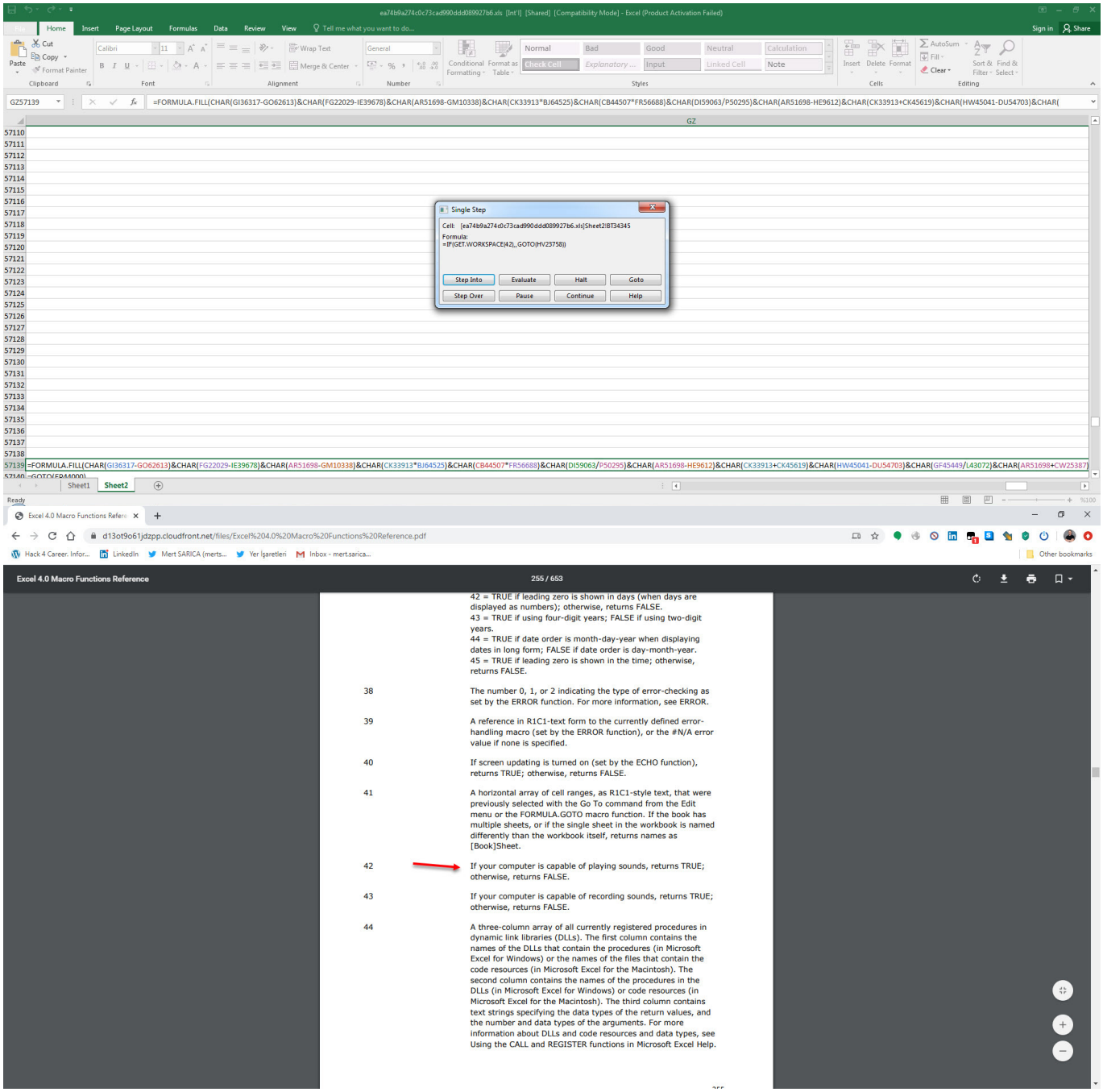

Excel 4.0 Macro Functions Refere
d13ot9o61jdzpp.cloudfront.net/files/Excel%204.0%20Macro%20Functions%20Reference.pdf
Hack 4 Career. Infor...
LinkedIn
Mert SARICA (merts...
Yer işaretleri
Inbox - mert.sarica...
Other bookmarks
Excel 4.0 Macro Functions Reference
253 / 653
Page length
Logical value indicating whether to wait after printing each page (TRUE) or use continuous form feeding (FALSE)
Logical value indicating whether the printer has automatic line feeding (TRUE) or requires line feed characters (FALSE)
The number of the printer port
252
myOnlineTraininghub
31 If a currently running macro is in single step mode, returns TRUE; otherwise, returns FALSE.
32 The current location of Microsoft Excel as a complete path.
33 A horizontal array of the names in the New list, in the order they appear.
34 A horizontal array of template files (with complete paths) in the New list, in the order they appear (returns the names of custom template files and the #N/A error value for built-in document types).
35 If a macro is paused, returns TRUE; FALSE otherwise.
36 If the Allow Cell Drag And Drop check box is selected in the Edit tab of the Options dialog box that appears when you click the Options command on the Tools menu, returns TRUE; otherwise, returns FALSE.
37 A 45-item horizontal array of the items related to country versions and settings. Use the following macro formula to return a specific item, where number is a number in the list below:
INDEX(GET.WORKSPACE(37), number)


ea74b9a274c0c73cad990ddd089927b6.xls [Int'l] [Shared] [Compatibility Mode] - Excel (Product Activation Failed)
AT41111
=FORMULA.FILL(CHAR(FG22029*BY34896)&CHAR(CB44508-IP29128)&(
=IF(GET.WORKSPACE(31),GOTO(HV23758),)
=GOTO(AY23948)
Sheet1
Sheet2
Ready
SUM
=FORMULA.FILL(CHAR(FG22029*BY34896)&CHAR(CB44508-IP29128)&CHAR(AR51699-IS46266)&CHAR(CB44508+GT49916)&CHAR(GI36317/BZ3699)&CHAR(HW45042*CK53811)&CHAR(DI59064+A65374)&CHAR(AR51699-AS62385)&CHAR(CB44508/D15017)&CHAR(FG22029/
CHAR(number)
Edit

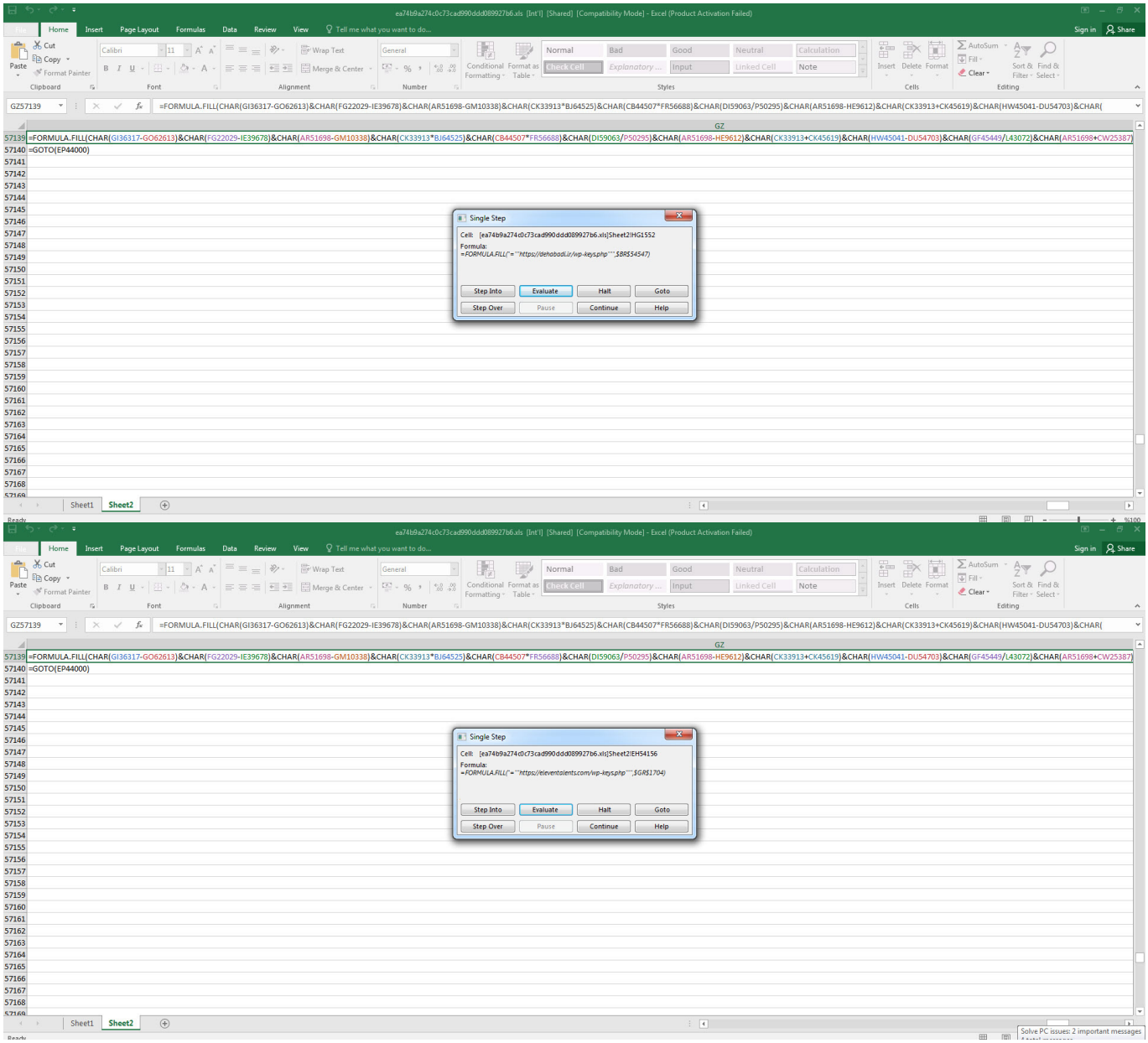

ea74b9a274c0c73cad990ddd089927b6.xls [Int'l] [Shared] [Compatibility Mode] - Excel (Product Activation Failed)
AU41111
AT
AU
AV
AW
AX
AY
AZ
BA
BB
BC
BD
41104 =GOTO(AY23948)
41105 =IF(GET.WORKSPACE(31),GOTO(HV23758),)
Sheet1
Sheet2


ea74b9a274c0c73cad990ddd089927b6.xls [Int'l] [Shared] [Compatibility Mode] - Excel (Product Activation Failed)
GZ57139
=FORMULA.FILL(CHAR(GI36317-GO62613)&CHAR(FG22029-IE39678)&CHAR(AR51698-GM10338)&CHAR(CK33913*BJ64525)&CHAR(CB44507*FR56688)&CHAR(DI59063/P50295)&CHAR(AR51698-HE9612)&CHAR(CK33913+CK45619)&CHAR(HW45041-DU54703)&CHAR(
GZ
Single Step
Cell: [ea74b9a274c0c73cad990ddd089927b6.xls]Sheet2!AT45304
Formula:
=IF(GET.WORKSPACE(19),,GOTO(HV23758))
Step Into
Evaluate
Halt
Goto
Step Over
Pause
Continue
Help
57139 =FORMULA.FILL(CHAR(GI36317-GO62613)&CHAR(FG22029-IE39678)&CHAR(AR51698-GM10338)&CHAR(CK33913*BJ64525)&CHAR(CB44507*FR56688)&CHAR(DI59063/P50295)&CHAR(AR51698-HE9612)&CHAR(CK33913+CK45619)&CHAR(HW45041-DU54703)&CHAR(GF45449/L43072)&CHAR(AR51698+CW25387)
Sheet1
Sheet2
Ready


Excel 4.0 Macro Functions Refere
d13ot9o61jdzpp.cloudfront.net/files/Excel%204.0%20Macro%20Functions%20Reference.pdf
4 = Data Entry
5 = Unused
6 = Copy and Data Entry
7 = Cut and Data Entry
If no special mode is set, returns 0.
11 X position of the Microsoft Excel workspace window, measured in points from the left edge of the screen to the left edge of the window. In Microsoft Excel for the Macintosh, always returns 0.
12 Y position of the Microsoft Excel workspace window, measured in points from the top edge of the screen to the top edge of the window. In Microsoft Excel for the Macintosh, always returns 0.
13 Usable workspace width, in points.
14 Usable workspace height, in points.
15 Number indicating maximized or minimized status of Microsoft Excel:
1 = Neither
2 = Minimized
3 = Maximized
Microsoft Excel for the Macintosh always returns 3.
16 Amount of memory free (in kilobytes).
17 Total memory available to Microsoft Excel (in kilobytes).
18 If a math coprocessor is present, returns TRUE; otherwise, returns FALSE.
19 If a mouse is present, returns TRUE; otherwise, returns FALSE. In Microsoft Excel for the Macintosh, always returns TRUE.
20 If a group is present in the workspace, returns a horizontal array of sheets in the group; otherwise returns the #N/A error value.

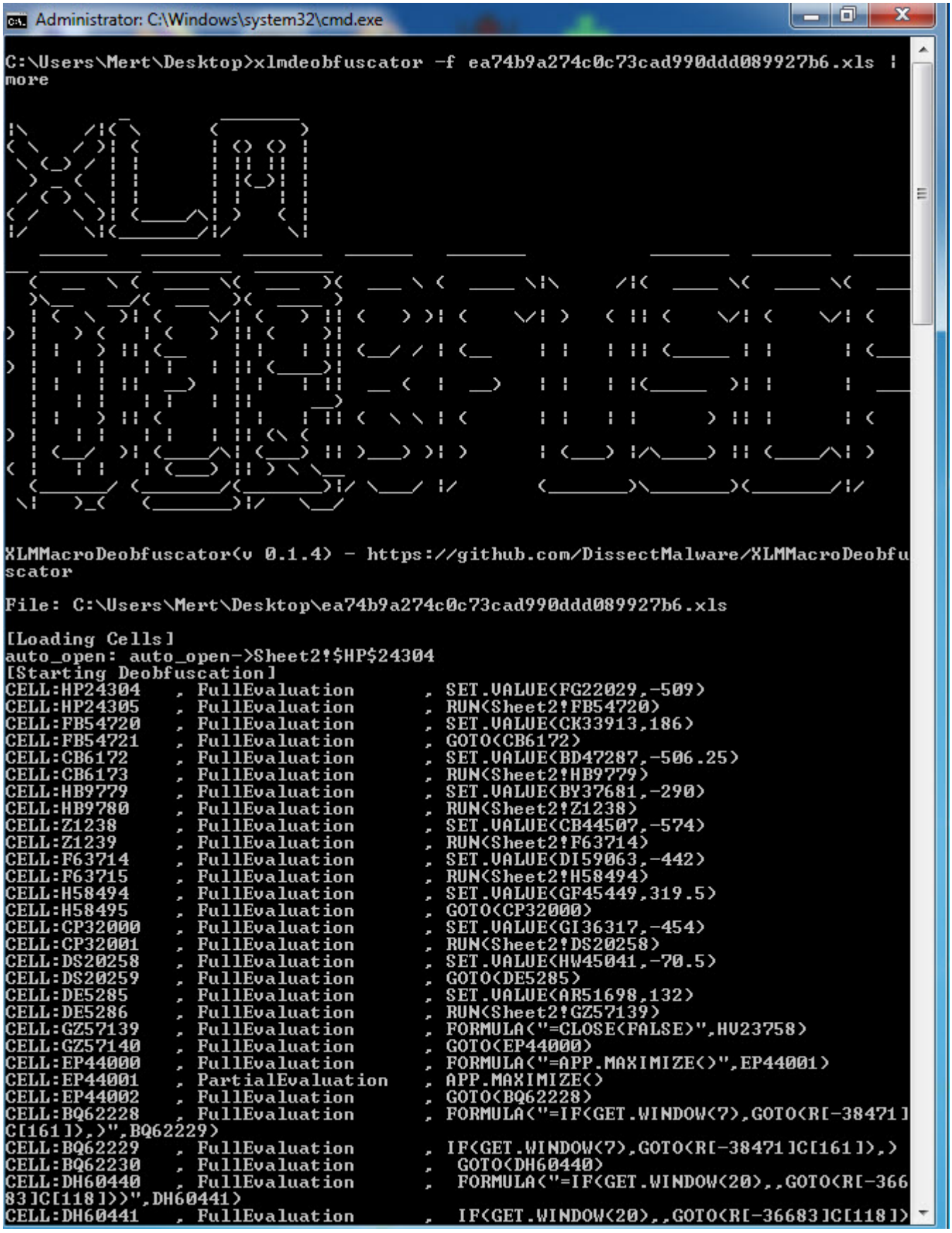


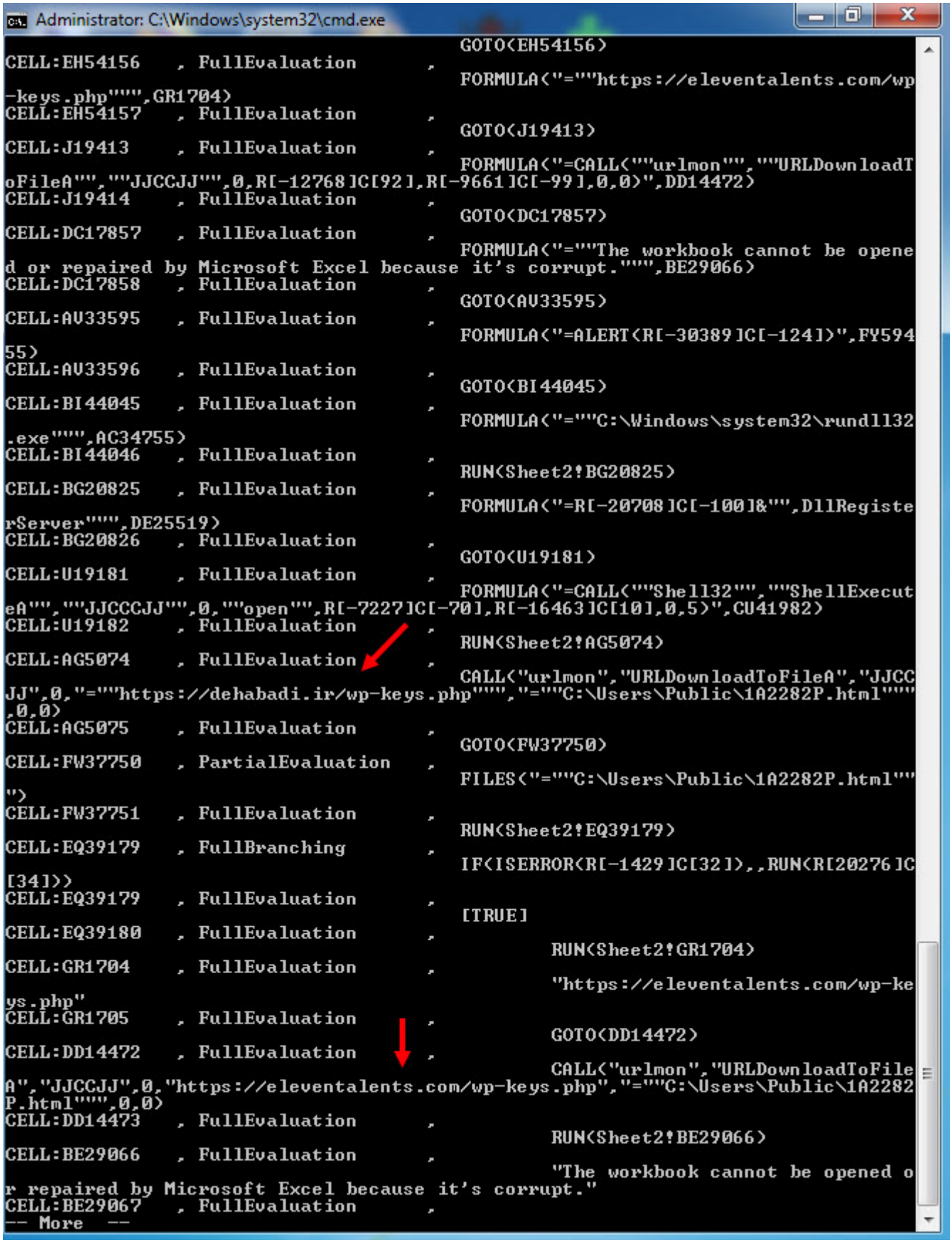
As I continued debugging, I noticed that the macro attempts to connect to https://docs.microsoft.com/en-us/officeupdates/office-msi-non-security-updates to check for internet connection and stops running if it encounters an error. I also observed that macro checks for permission for macro usage via registry, After that it try to contact with https://dehabadi[.]ir/wp-keys[.]php and https://eleventalents[.]com/wp-keys[.]php. Although these addresses were not active during my analysis, my research led me to suspect that these addresses are command and control servers associated with the Zloader malware. Even though I couldn’t continue my analysis, my aim was reached successfully by revealing these addresses.



With this explanation you have a basic understanding of the subject matter, now you can use a tool such as XLMMacroDeobfuscator to quickly solve the hidden XLM macro and save time. With this article I hope to provide insight for analysts who want to analyze XLM macros.

Hope to see you in the following articles.


Administrator: C:\Windows\system32\cmd.exe
C:\Users\Mert\Desktop>xlmdeobfuscator -f ea74b9a274c0c73cad990ddd089927b6.xls | more
XLMMacroDeobfuscator(v 0.1.4) - https://github.com/DissectMalware/XLMMacroDeobfuscator
File: C:\Users\Mert\Desktop\ea74b9a274c0c73cad990ddd089927b6.xls
[Loading Cells]
auto_open: auto_open->Sheet2!HP24304
[Starting Deobfuscation]
CELL:HP24304 , FullEvaluation , SET.VALUE(FG22029,-509)
CELL:HP24305 , FullEvaluation , RUN(Sheet2!FB54720)
CELL:FB54720 , FullEvaluation , SET.VALUE(CK33913,186)
CELL:FB54721 , FullEvaluation , GOTO(CB6172)
CELL:CB6172 , FullEvaluation , SET.VALUE(BD47287,-506.25)
CELL:CB6173 , FullEvaluation , RUN(Sheet2!HB9779)
CELL:HB9779 , FullEvaluation , SET.VALUE(BY37681,-290)
CELL:HB9780 , FullEvaluation , RUN(Sheet2!Z1238)
CELL:Z1238 , FullEvaluation , SET.VALUE(CB44507,-574)
CELL:Z1239 , FullEvaluation , RUN(Sheet2!F63714)
CELL:F63714 , FullEvaluation , SET.VALUE(DI59063,-442)
CELL:F63715 , FullEvaluation , RUN(Sheet2!H58494)
CELL:H58494 , FullEvaluation , SET.VALUE(GF45449,319.5)
CELL:H58495 , FullEvaluation , GOTO(CP32000)
CELL:CP32000 , FullEvaluation , SET.VALUE(GI36317,-454)
CELL:CP32001 , FullEvaluation , RUN(Sheet2!DS20258)
CELL:DS20258 , FullEvaluation , SET.VALUE(HW45041,-70.5)
CELL:DS20259 , FullEvaluation , GOTO(DE5285)
CELL:DE5285 , FullEvaluation , SET.VALUE(AR51698,132)
CELL:DE5286 , FullEvaluation , RUN(Sheet2!GZ57139)
CELL:GZ57139 , FullEvaluation , FORMULA("=CLOSE(FALSE)",HV23758)
CELL:GZ57140 , FullEvaluation , GOTO(EP44000)
CELL:EP44000 , FullEvaluation , FORMULA("=APP.MAXIMIZE()",EP44001)
CELL:EP44001 , PartialEvaluation , APP.MAXIMIZE()
CELL:EP44002 , FullEvaluation , GOTO(BQ62228)
CELL:BQ62228 , FullEvaluation , FORMULA("=IF(GET.WINDOW(7),GOTO(R[-38471]C[161]),)",BQ62229)
CELL:BQ62229 , FullEvaluation , IF(GET.WINDOW(7),GOTO(R[-38471]C[161]),)
CELL:BQ62230 , FullEvaluation , GOTO(DH60440)
CELL:DH60440 , FullEvaluation , FORMULA("=IF(GET.WINDOW(20),,GOTO(R[-36683]C[118]))",DH60441)
CELL:DH60441 , FullEvaluation , IF(GET.WINDOW(20),,GOTO(R[-36683]C[118])



Note: For those looking for more resources on XLM macro analysis, I recommend looking at these articles (#1, #2, #3, #4, #5) as well. These articles will give more information about the analysis of XLM macros and methods that you can use.