Newly Observed Ursnif Variant Employs Malicious TLS Callback Technique to Achieve Process Injection

fireeye.com/blog/threat-research/2017/11/ursnif-variant-malicious-tls-callback-technique.html



Threat Research Blog

November 28, 2017 | by <u>Abhay Vaish</u>, <u>Sandor Nemes</u> <u>Malware</u> <u>Malware Analysis</u>

Introduction

TLS (Thread Local Storage) callbacks are provided by the Windows operating system to support additional initialization and termination for per-thread data structures.

As <u>previously reported</u>, malicious TLS callbacks, as an anti-analysis trick, have been observed for quite some time and can allow for PE files to include malicious TLS callback functions to be executed prior to the AddressOfEntryPoint field (the normal start of intended execution) in the PE header. In essence, unsuspecting analysts and automated security tools can miss the actual entry point to malcode if they do not account for it in the beginning of their analysis and insert a breakpoint on the regular offset pointed to by AddressOfEntryPoint.

We recently came across a Ursnif/Gozi-ISFB sample that manipulated TLS callbacks while injecting to child process. Though many of the malware binaries (or their packers) use some variation of GetThreadContext/SetThreadContext or CreateRemoteThread Windows API functions

to change the entry point of the remote process during injection, this sample (and the related cluster) is using a relatively lesser-known stealth technique. This little deviation from the standard textbook approach may cause some generic unpackers or tools to break following the execution flow, if they do not account for the technique.

Distribution

Since early 2017, we have regularly observed the abuse of compromised Sharepoint accounts to host malicious payloads, with distribution of URIs via spam emails. Some of the major campaigns we've observed involve distributing Dridex within the UK and Ursnif/Gozi-ISFB in Australia. The recently observed Ursnif variant discussed in this post was discovered via a spam email. A sample lure can been seen in Figure 1.

From Subject Reply to To	a land via DocuSign <@docusign.s1email.com>☆ Your MYOB Supply Order a main a model of the second of the	08:54
	myob	
	I am sending you this Supply Order for your electronic signature, please review and electronically sign by following the Review Document link below.	
	David Conlan sent you a document to review and sign. REVIEW DOCUMENT	
	Regards, @myob.com	
	Powered by Docu Sign	
	Australian Sales and Service Agreement Click here to view	

Figure 1: Malicious email lure distributing Ursnif

After clicking on the "REVIEW DOCUMENT" button, the malware downloads a ZIP file named *YourMYOBSupply_Order.zip* from the following location:

https://eacg1-

my.sharepoint.com/personal/steve_robson_eaconsultinggroup_com/_layouts/15/download.aspx? docid=<*redacted*>&authkey=<*redacted*>

The ZIP file contains a malicious JavaScript file that, when executed, will download and execute the Ursnif/Gozi-ISFB payload.

The activities of the distribution are difficult to identify within an organization's normal network activity because the command and control (C2) server of this payload communicates over HTTPS and the compromised Sharepoint accounts being used also communicate over HTTPS.

Variant Analysis

On execution, the observed sample (MD5: 13794d1d8e87c69119237256ef068043) tries to create a child process named *svchost.exe* (using the *svchost.exe* file from the System32 folder) using the CreateProcessW API function in suspended mode.

Next, for process hollowing of *svchost.exe*, the malware creates a section object and maps the section using ZwMapViewOfSection. It uses the memset function to fill the mapped section with zeroes, and then leverages memcpy to copy the unpacked DLL to that region. The malware then resolves three lower level API functions by walking the ntdll.dll module.

The malware then constructs its entry shellcode into a newly mapped region in memory.

In an effort to manipulate and identify the mapped sections of the child process, it reads out the PEB structure of the process using a call to ZwReadVirtualMemory.

The malware will then change protection permissions of the PE header of the child process to enable write access to that page. It then uses a call to ZwWriteVirtualMemory to write 18 bytes of buffer at offset 0x40 from the start of *svchost.exe* process executable in the target child process. The malware then cleverly changes the region protection back to "read only" to avoid suspicion.

Again, it repeats the procedure of changing protections for the PE image of *svchost.exe* to write 8 bytes at an offset of 0x198 bytes from the start of the process executable.

The Stealthy Tweak

This buffer, when correctly placed at the offset, will represent the TLS directory offset for the process because offset 0x198 is the location of the TLS directory in PE executable, and the next DWORD represents the size of the directory (seen in Figure 2). Notice how the malware writes the offset 0x40 for directory and the size 0x18 bytes in an effort to point to the buffer it had already crafted at offset 0x40 with size 0x18 bytes.

TLS Directory RVA	00000198	Dword
TLS Directory Size	0000019C	Dword

Figure 2: TLS directory location and size

The TLS directory structure, when used to parse out that buffer of 0x18 bytes, points to an offset containing a list of pointers representing AddressOfCallBacks (see Figure 3).

struct	_IMAGE_TLS	DIRECTORY {
0x00	DWORD	StartAddressOfRawData;
0x04	DWORD	EndAddressOfRawData;
0x08	LPDWORD	AddressOfIndex;
0x0c	PIMAGE_TL	S_CALLBACK *AddressOfCallBacks;
0x10	DWORD	SizeOfZeroFill;
0x14	DWORD	Characteristics;
};		

Figure 3: TLS directory structure with pointers

If we take a look at offset 0xe058, it points to the list of AddressOfCallBacks (Figure 4), and if we go to the offset 0xe058 in memory we are pointed to the only callback address at offset 0xe068 - which is in fact the actual entry point code (Figure 5).

eral Statistics Perfor	mance Thread	s Token M	odules	Memory	Env	ironme	nt H	andle	s C	omme	ant									
Hide free regions																		String	s	Refresh
ase Address	Туре		Size	Prot	ect	Use								То	tal WS	Pri	vate WS	Share	able WS	Shared V
▷ 0x10000	Private		128 kE	B RW											8 kB		8 kB			
▷ 0x30000	Mapped		16 kE	B R											16 kB				16 kB	16
▷ 0x40000	Mapped		4 kE	B R											4 kB				4 kB	4
▷ 0x50000	Private		4 kE	B RW											4 kB		4 kB			
4 0x60000	Mapped		60 kE	8 RWX											60 kB				60 kB	
0x60000	Mapped: Commi	t	60 kE	RWX											60 kB				60 kB	
▷ 0xe0000	Private (<i>e</i>													
4 0xc00000	Image	svchost	.exe (308	0) (0x6	- 0000	0x6f0	00)											• X	🖌 🖥 🔚	12
0xc00000	Image: Commit																			
0xc01000	Image: Commit	0000dfd	0 00 00	00 0	0 00	00 0	0 00	00	00	00 0	0 00	00	00 (00.	• • • • • •	• • • •		^	8 kB	4
0xc05000	Image: Commit	0000dfe			0 00	00 0	0 00	00	00	0 00	0 00	00	00 (. 00	•••••	• • • •			₽ kB	4
0xc06000	Image: Commit	00000011	00000 b823		7 00	00 0		84	22	-f 7	7 00	00	00 0	. 00		···			₽ kB	4
▷ 0x77c90000	Image	0000e01	18 51	E cd 7	7 00	00 0	0 00	00	00	00 0	0 00	00	00 0	00.	.w				2 kB	8
▷ 0x77ed0000	Image	0000e02	0 00 00	0 00 0	0 00	00 0	0 00	00	00	00 0	0 00	00	00 (00 .						
▷ 0x7ffb0000	Mapped	0000e03	0 00 00	060	0 00	00 0	0 00	00	00	0 00	0 00	00	00 (00.			Callb	back	2 kB	32
▷ 0x7ffd3000	Private	0000e04) 68 e1	L 06 0	0 00	00 0	0 00	00	eu	10 0	0 00	00	00 (00 h						
▷ 0x7ffdf000	Private	0000e05	0 00 00	0000	0 00	00 0		68	e0 (06 0	0 00	00	00 (00.	• • • • • •	.h. 2	adr	ess		
▷ 0x7ffe0000	Private	0000e06	0 00 0	UTTS	eta	Jxe	026	50			00 00	00	58 1	51.		.F	xQ			
		0000e07	153 68	5 36 3	0 95	II 3	0 41	10 ##	42	48 1 5a 5	0 0 0	06	01 0	. au	-03U	AH	u.			
		0000e09	0 87 04	1 24 0	3 59	58 0	2 00	00	e8 1	00 0	0 00	00	58 1	51 .	.s. YX.	ו••				
		0000e0a	48 83	3 e8 3	5 8b	c8 f	f 30	fO	87	48 1	8 85	c9	75 3	20 H		0н	u			
	1	0000e0b	52 41	L 50 4	1 51	55 4	8 8b	ec	48	83 e	c 38	48	8b !	50 R	AFAQUH	н.	.8H.P		-	•
		0000e0c	08 48	8 8b 4	8 10	ff d	12 48	8b	e5 !	5d 4	1 59	41	58 \$	5a .	н.н	н]2	AYAXZ			

Figure 4: Offset 0xe058

neral Statistics Pe	rformance Thread	ls Token Module	s Men	nory E	nvironr	nent	Handle	es (Comm	ent								
Hide free regions																Strings] [Refresh
Base Address	Туре		Size P	Protect	. Use	2		_	_	_	_		т	otal WS	Private WS	Sharea	ble WS	Shared
▷ 0x10000	Private	12	8 kB F	RW										8 kB	8 kB			
▷ 0x30000	Mapped	1	6 kB F	2										16 kB			16 kB	16
⊳ 0x40000	Mapped		4kB R	2										4 k8			4 kB	4
▷ 0x50000	Private		4kB F	RW										4 kB	4 kB			
4 0x60000	Mapped	6	0 kB F	RWX										60 kB			60 kB	
0x60000	Mapped: Comm	it 6	0 kB R	RWX										60 kB			60 kB	
▷ 0xe0000	Private (••••		1 61												
4 0xc00000	Image	svchost.exe (3080) (0x60000	- 0x6f	(000										• ×	i kB	1
0xc00000	Image: Commit																	
0xc01000	Image: Commit	0000dfd0 00	00 0	0 00 0	0 00	00 0	00 00	00	00 (00 00	00 0	00	00			~	8 kB	4
0xc05000	Image: Commit	0000dfe0 00	00 0	0 00 0	0 00	00 0	00 00	00	00 0	00 00	00 0	00	00	• • • • • • • • •	• • • • • • • • •		₽ kB	4
0xc06000	Image: Commit	0000dff0 00	00 0	0 00 0	0 00	00 0	00 00	00	00 0		00 0	00	00				₽ kB	
≥ 0x77c90000	Image	0000e010 18	22 C	1 77 0	0 00	00 0	0 00	22	CI I			00	00	· · · W · · · ·	W		2 kB	8
▷ 0x77ed0000	Image	0000e020 00	00 0	0 00 0	0 00	00 0	0 00	00	00 0		000	00	00					
▷ 0x7ffb0000	Mapped	0000e030 00	00 0	6 00 0	0 00	00 0	00 00	00	00 0	00 00	00	00	00				2 kB	33
> 0x7ffd3000	Private	0000e040 68	e1 0	6 00 0	0 00	00 0	00 00	e0	06 (00 00	00 0	00	00	h				
▷ 0x7ffdf000	Private	0000e050 00	00 0	0 00 0	0 00	00 0	00 68	e0	06 (00 00	00 (00	00		h			
▷ 0x7ffe0000	Private	0000e060 00	00 0	0 00 0	0 00	00 0	00 50	e8		00 00	00 0	58	51		FXQ			
		0000e070 83	e8 3	6 33 c	9 ff	30 4	41 f0	87	48 :	18 85	5 C9	75	0b		Hu.			
		0000e080 52	80 4	8 10 8	D 50	08 5		d2	5a :	59 el	5 06	91	59		ZYY			
		0000e0a0 42	83 e	4 C3 3 8 35 8		- C2 (30 F0	87	48 1		5 00	75	20		H 11			
	1	0000e0b0 52	41 5	0 41 5		48 8	Bb ec		83	ec 38	3 48		50		.H8H.P		-	
	٩	0000e0c0 08	48 8	b 48 1	0 ff	d2 4	18 8b		5d 4	41 59	9 41	58	5a]AYAXZ		_	
		0000e0d0 58	59 8		4 01			ff	25 !	58 91			ff	XYt.F.				

Figure 5: Offset 0xe068 in memory

Finally, the malware unmaps the view using ZwUnmapViewOfSection and calls ResumeThread to begin malicious execution of its injected process (from the injected TLS callback address instead of the regular AddressOfEntryPoint listed in the PE header). Hence, the execution will first land at the injected TLS callback (see Figure 6).

0006E068	50	PUSH EAX 🛌
0006E069	<mark>28</mark> 00000000	CALL 0006E06E
0006E06E	58	POP ERX
00065065	51	
00065070	83E8 36 99C9	See Elores Actual Entry Point
000062073	55C7 EE90	PHEN DHORD PTP DE LEONI
0000E013	41	INC FCX
0006E078	F0:8748 18	LOCK XCHG DWORD PTR DS:[EAX+18].ECX LOCK prefix
0006E07C	8509	TEST ECX, ECX
0006E07E	75 ØB	JNZ SHORT 0006E08B
0006E080	52	PUSH_EDX
0006E081	8B48 10	MOV ECX, DWORD PTR DS: [EAX+10]
0006E084	8850 08	MOV EDX, DWORD PTR DS: [ERX+8]
0006E087	51	PUSH EUX
000052088		UHLL EUA

Figure 6: Actual entry point

Impact

The <u>leaked source code</u> of Ursnif/Gozi-ISFB used the standard DIIMain call entry point to initialize the injected DLL image and execute its entry (see Figure 7).

702	L.	
170		// Calling DLL entry point
171		<pre>pDllMain = (FUNC_DLL_MAIN)(ImageBase + Pe->OptionalHeader.AddressOfEntryPoint);</pre>
172		Result = (pDllMain)(ImageBase, DLL_PROCESS_ATTACH, &LdrCtx->AdContext);
173	}	<pre>// if (NT SUCCESS(ntStatus))</pre>

Figure 7: DIIMain call used in leaked Ursnif source code

This newer variant shows that actors are not only modifying the malware to evade signatures, they are also equipping them with stealthier techniques. Unaware debugging environments or detection frameworks can potentially miss the actual hidden TLS callback entry point, allowing the malware to perform its malicious activities under the hood.

Indicators of Compromise

Filename :YourMYOBSupply_Order.zip MD5 : f6ee68d03f3958785fce45a1b4f590b4 SHA256 : 772bc1ae314dcea525789bc7dc5b41f2d4358b755ec221d783ca79b5555f22ce

Filename : YourMYOBSupply_Order.js MD5 : c9f18579a269b8c28684b827079be52b SHA256 : 9f7413a57595ffe33ca320df26231d30a521596ef47fb3e3ed54af1a95609132

Filename : download[1].aspx MD5 : 13794d1d8e87c69119237256ef068043 SHA256 : e498b56833da8c0170ffba4b8bcd04f85b99f9c892e20712d6c8e3ff711fa66c

Previous Post Next Post