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 Abhay Vaish, Sandor Nemes
Malware
Malware Analysis

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 previously reported, 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.

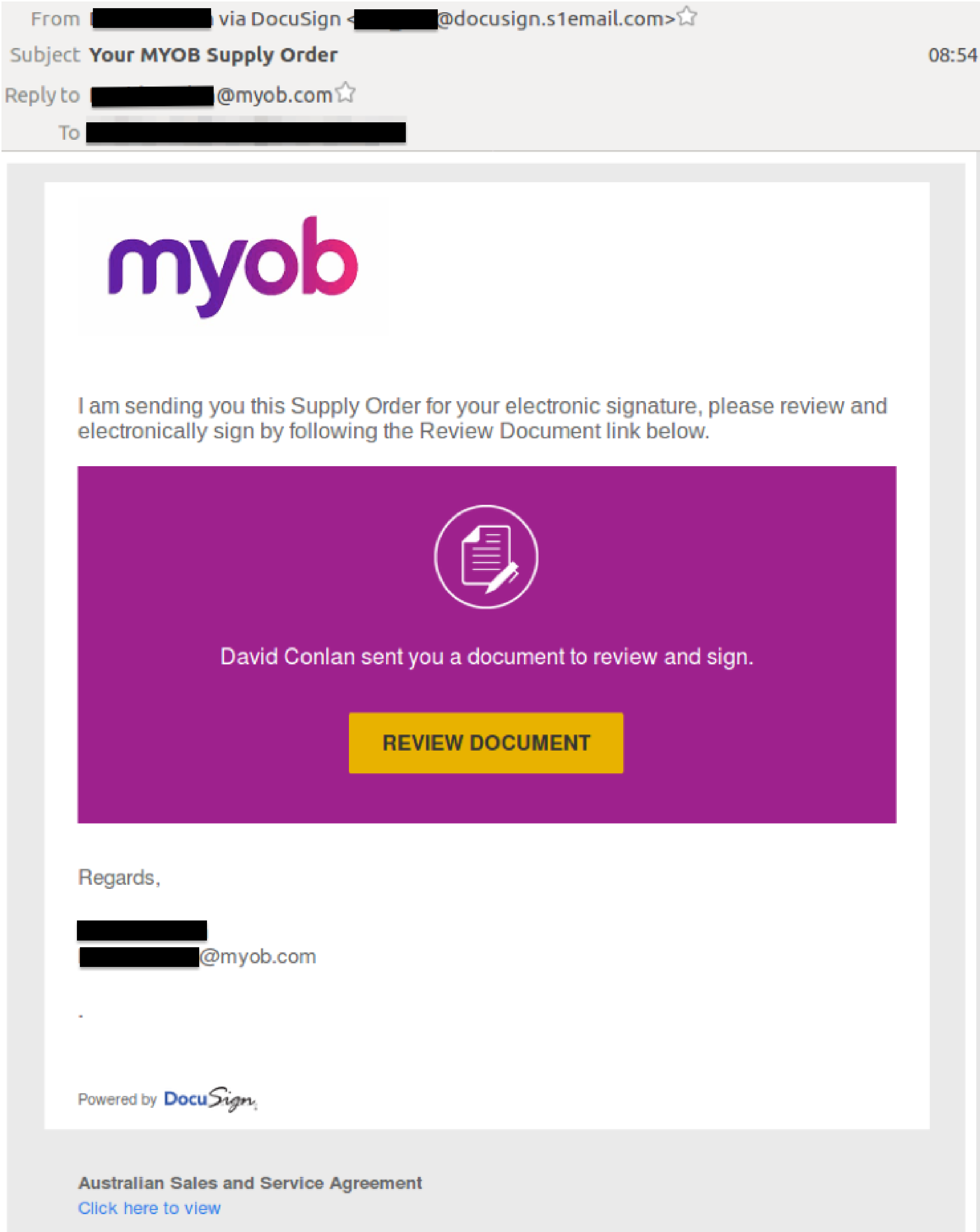

Figure 1: Malicious email lure distributing Ursnif

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_TLS_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).

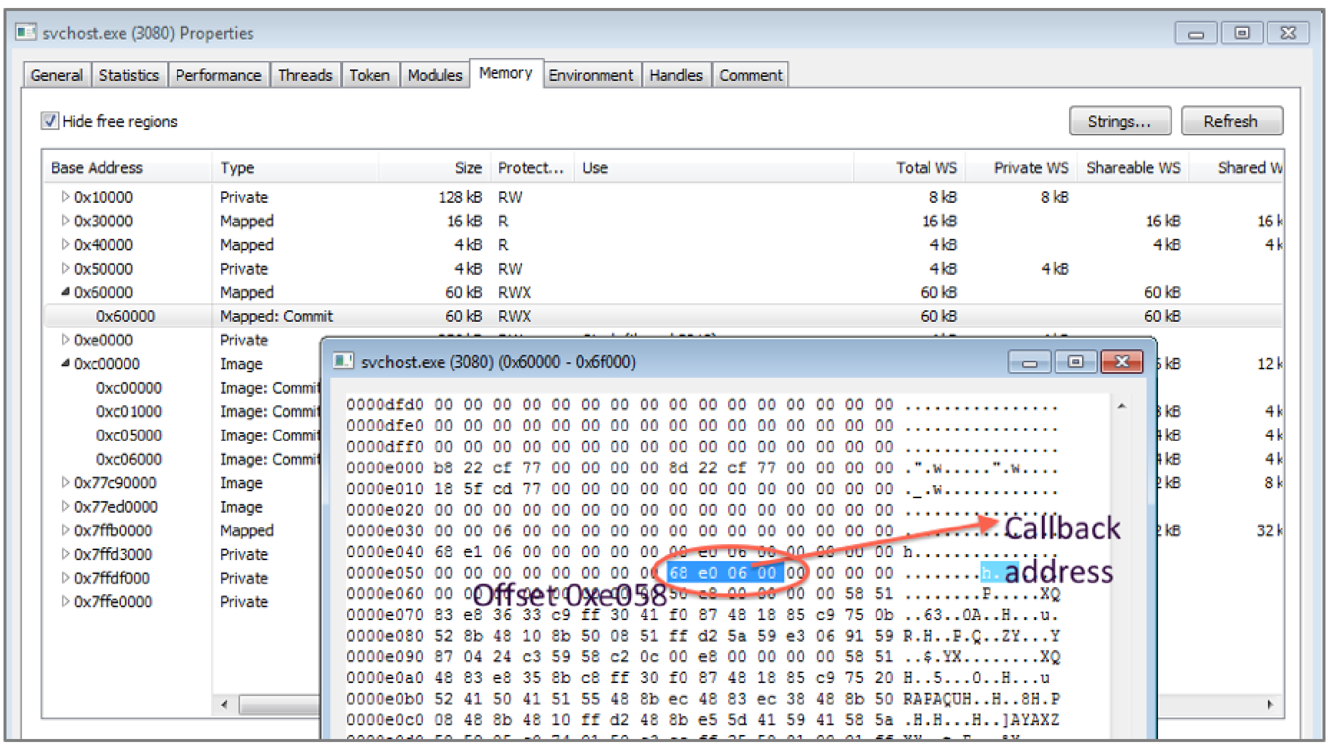

Figure 4: Offset 0xe058

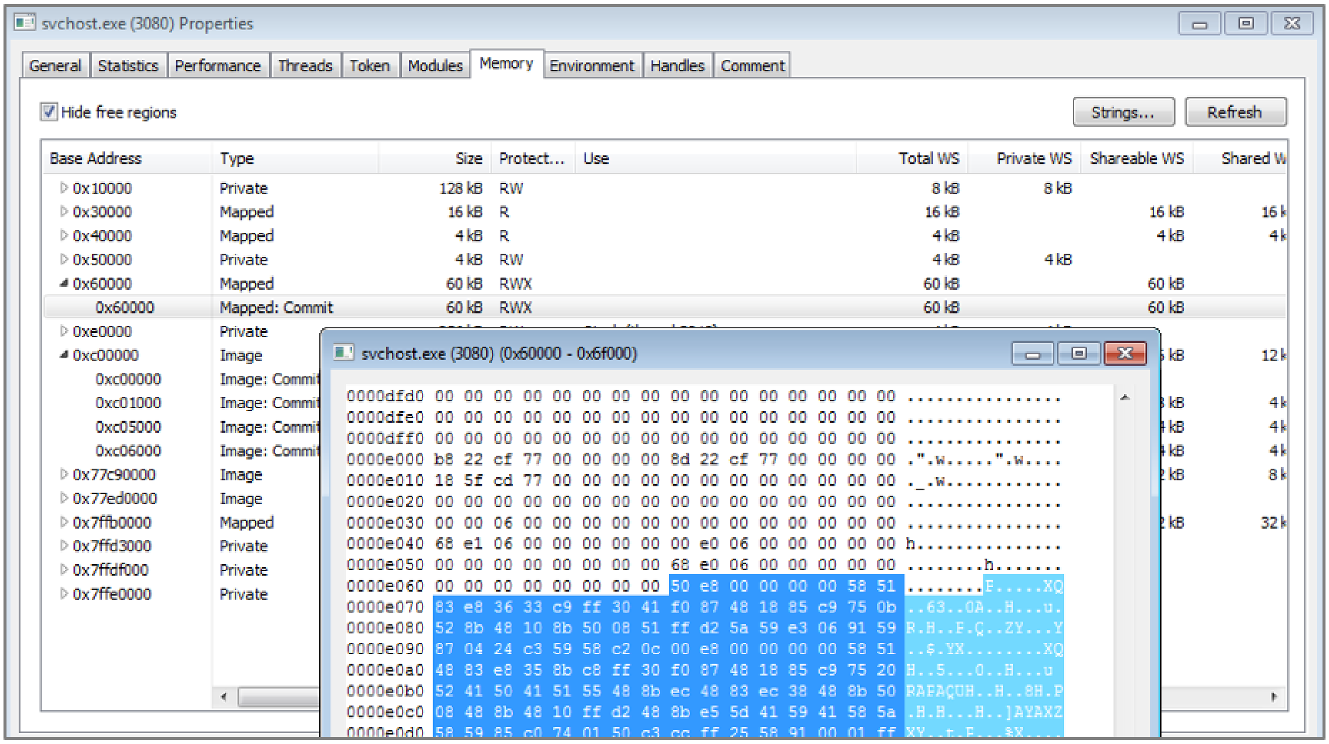

Figure 5: Offset 0xe068 in memory

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).

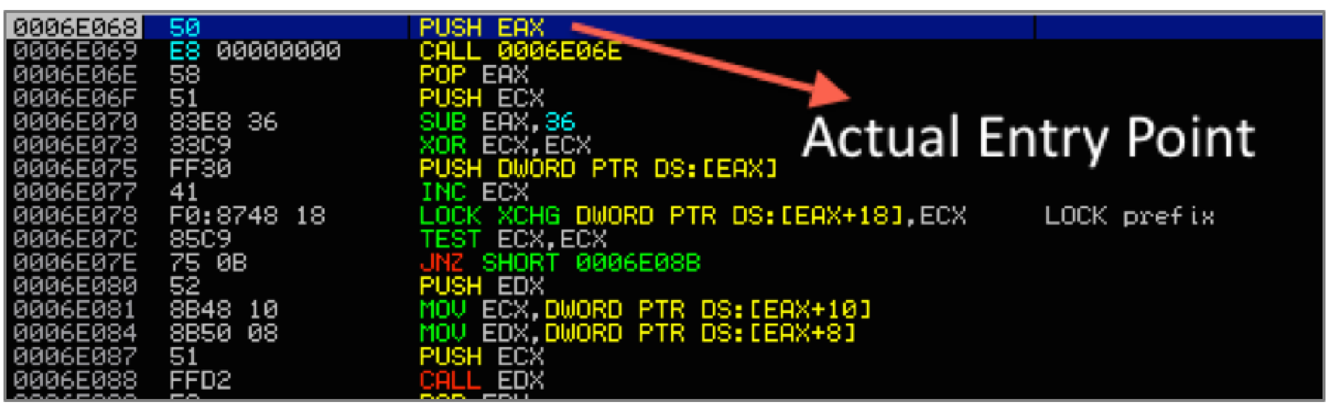

Figure 6: Actual entry point

Impact

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

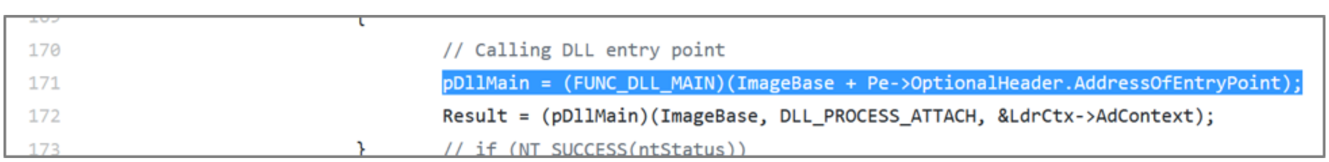

Figure 7: DllMain call used in leaked Ursnif source code

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