A Relook at the TerraLoader Dropper DLL

By Jason Reaves

Published: 2021-04-09 · Archived: 2026-04-05 14:37:52 UTC

Press enter or click to view image in full size

The malware author responsible for the TerraLoader, who is believed to be BadBullzVenom[4] after a disagreement between the actor and customer took place on an underground forum market place, appears to have been happily updating their delivery system again referred to as TerraLoader. An excellent report was posted online by Twitter user @Arkbird_SOLG [1,5] using a sample discovered by @malz_intel. The report was well done so this is mostly an addendum of my reverse engineering notes after analyzing the aforementioned sample in the report[2] where I primarily focus on the DLL that will build and deliver the javascript backdoor commonly referred to as 'more_eggs'.

Initial Overview

This is just a quick overview getting to the DLL portion, as aforementioned it is already covered in previous research[1].

Scriptlet:

```
MD5:    f93fdcf5856feb254a26547dc8c2b671
SHA1:   1c038a86b337cf72663da5b6780d34ba0fb1c0c9
```

```
SHA256:  000a5e63109b3c653d63d84d03fe474242b987bfadda9aeaa200653fd2155a31
```

Scriptlet decodes out a DOC file and a DLL, the doc file appears to be a distraction from the loading of the DLL.

```
Doc: 11e54f594949a4c7f13e85b4ac2cbd555200ac34e6d61b58a71fbcfccc0497cd
DLL: fb55b26b8edee2431b35d0e28df5f223510a15344ede400a2f5c04a0d45e6b77
```

String Encoding Updates

The string encoding is a bit different now, the dropper DLL portion has always utilized a simple XOR based routine for bruting out its own string encoding key while also leveraging various secondary routines to use the key once found(XOR, AES, RC4). The author has moved to using RC4 for bruting out the string encoding key which makes the entire process take longer now but has the added benefit of acting as a sleep routine as well.

The sample takes three pieces of data, data that is encrypted, a starting RC4 key string and the data that will be decrypted to check that the correct key has been found.

Encrypted data:

```
b8e0ce81bcd3023bf5e2b37d3e1c801d99bd3f7f408c87b7923673d7840797a024f78d62552de17aba3d10c4
```

Starting RC4 key:

```
SouXRE
```

Decrypted data:

```
b92fe2de81d94d05b0431c545221cfbc23ad2f47a9279a20498948b9d4b3ffdf4b22669a1eace7a3b14862d2
```

The bruting routine works by simple using an iterator starting at 0 and converting the integer value into a string, appending it to the starting key, decrypting the encrypted data and checking that the output matches the decrypted data. If it doesn’t match the iterator is incremented and the process starts over again until the match is found.

If you understand how encryption algorithms work then you can see a flaw in this from a static reverse engineering perspective we can actually recover enough of the keystream data to decode out most of the strings on board without needing to brute force the key.

```
encoded_data ^ decoded_data = partial RC4 XOR Keystream
```

Using this knowledge we can recover part of the RC4 XOR keystream:

```
01cf2c5f3d0a4f3e45a1af296c3d4fa1ba101038e9ab1d97dbbf3b6e50b4687f6fd5ebf84b8106d90b757216
```

Using that data we can decode out most of the strings from the sample which all line up with previously listed strings, the malware is basically designed to build out the more_eggs backdoor for dropping to disk and detonating.

Get Jason Reaves's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

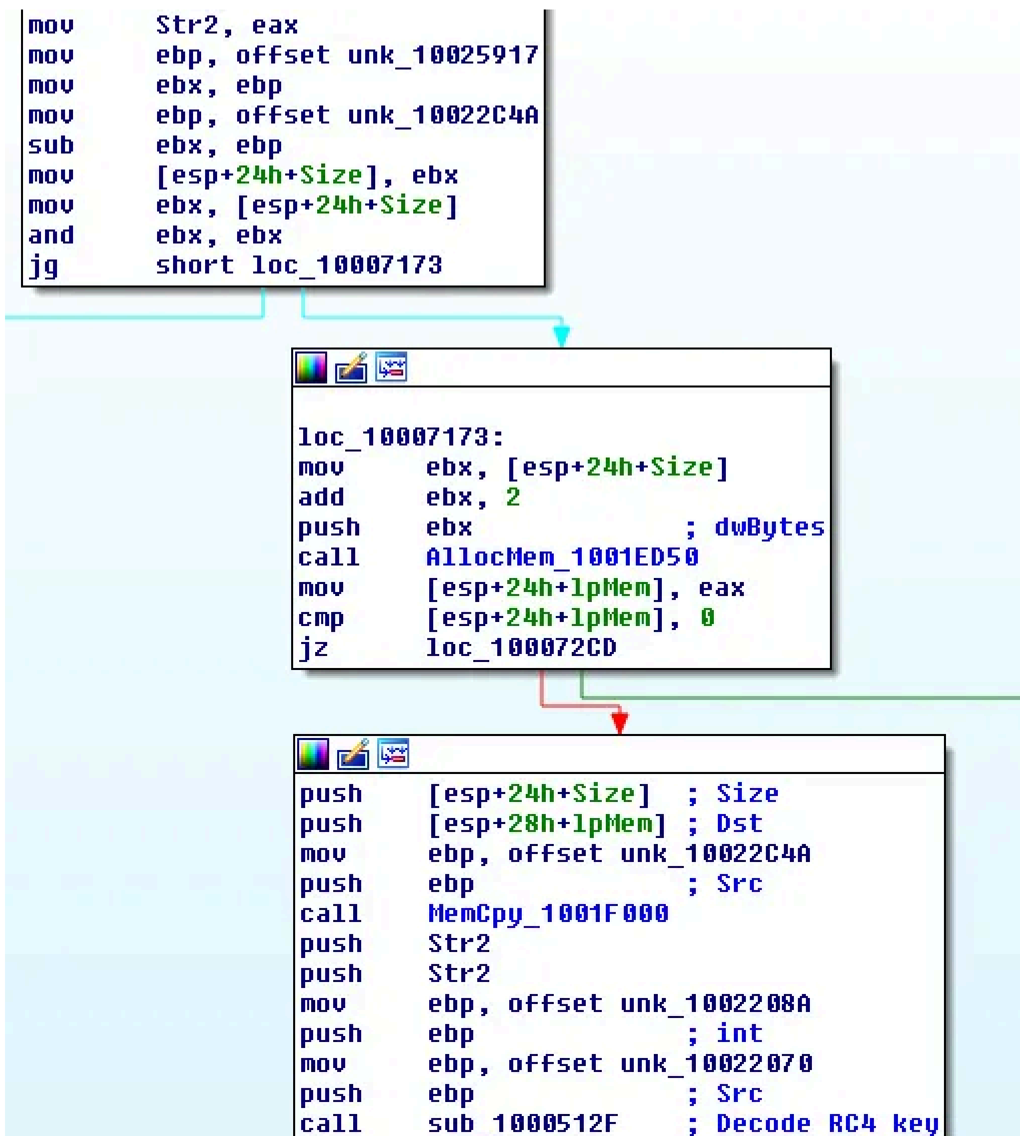
Partial strings for reference:

```
%APPDATA%
/B /e:jsCript
msxsl.exe
CED1712A510453BEE1F83F8AE7
a2service.exe
schtasks.exe
 /Create /TN
UserInitMprLogonScript
PROCESSOR_IDENTIFIER
.ComputerName +
COMPUTERNAME
FLAREVM
USERNAME
Notepad
```

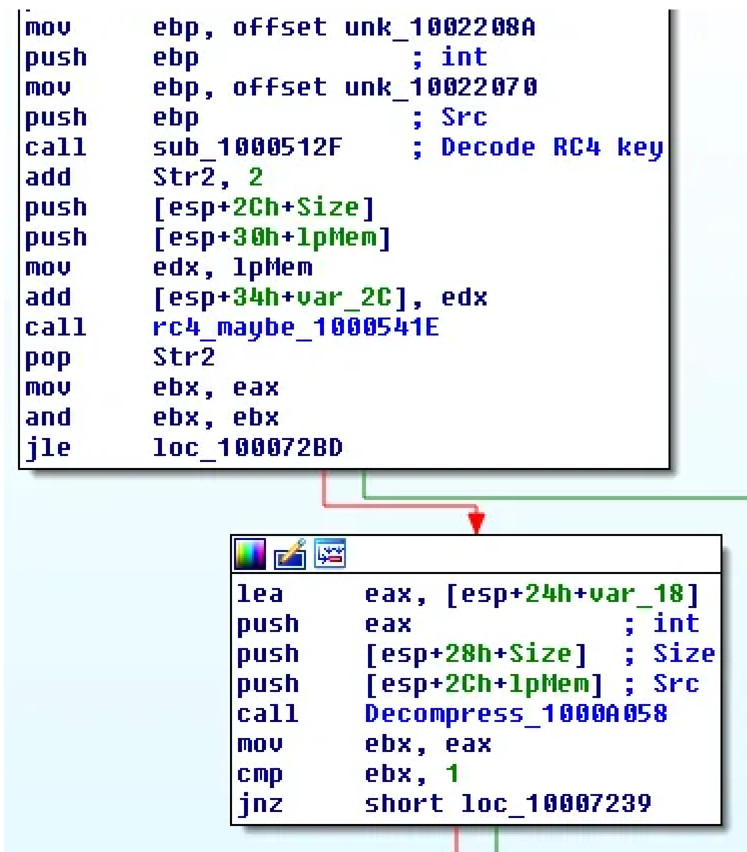
Data Encoding Updates

As mentioned by the earlier referenced blog post author this new backdoor is actually dropped by being encoded with the addition of computer based information added to the key, this makes it more difficult to recover the backdoor unless you know the computer name and processor information of the system it was dropped on.

We can see evidence of this in the partial strings above showing the 'PROCESSOR_IDENTIFIER' and 'COMPUTERNAME' along with '.ComputerName +' which will be used by parts of the wrapper on the backdoor to decode out the backdoor. If the Dropper DLL is responsible for this piece then the backdoor must exist somewhere in the DLL:



In the screenshot above we can see a large block of data has the size calculated, memory allocated, copied into the new memory and then the string ‘CED1712A510453BEE1F83F8AE7’ is decrypted. Right after this a call to the RC4 routine takes place:



After calling RC4, labeled ‘RC4_maybe’ because this picture was taken while I was still mapping out the sample, a call to an onboard DEFLATE routine takes place. Let’s check what is decoded:

```
>>> key = 'CED1712A510453BEE1F83F8AE7'
>>> from Crypto.Cipher import ARC4
>>> rc4 = ARC4.new(key)
>>> t = rc4.decrypt(data)
>>> import zlib
>>> t[:50]
'\xed}kw#7\xae\xe0g\xe7\x9c\xfc\x87\x8a\xee\x9d\xb44\xed\x96\xeb\xa1g\x94\xce\xac^N\xf7\xa6\x1f\xbe\
>>> t2 = zlib.decompress(t,-15)
>>> t2[:100]
'var BV = "6.6b";\r\nvar Gate = "https://d27qdop2sa027t.cloudfront.net/spmar/d9264";\r\nvar hit_each
```

Later in the sample we can also recover the decoding of MSXSL EXE file as well:

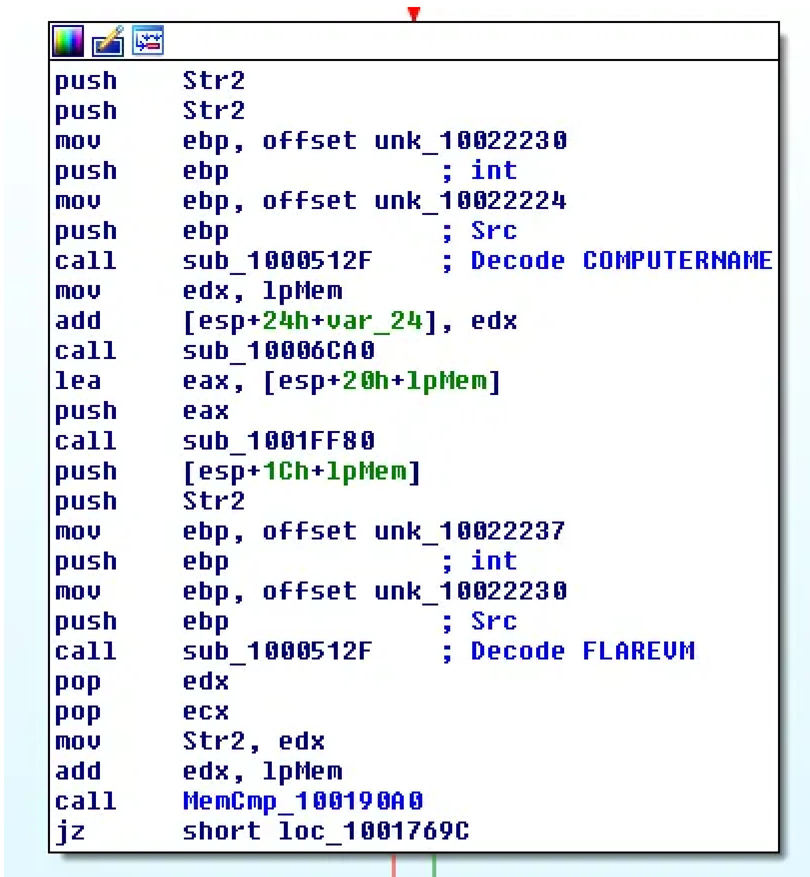
```
>>> key = 'CED1712A510453BEE1F83F8AE7'
>>> from Crypto.Cipher import ARC4
>>> rc4 = ARC4.new(key)
>>> t = rc4.decrypt(data1)
>>> import zlib
>>> t2 = zlib.decompress(t,-15)
```

```
>>> t2[:100]
'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00\xb8\x00\x00\x00\x00\x00\x00\x00@\x00\x00
```

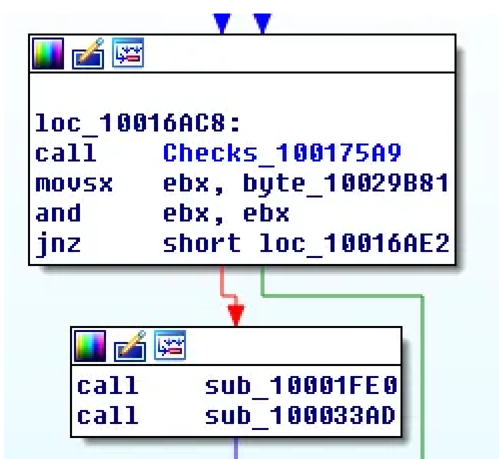
Also a part of the XML for task scheduling:


```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Author>SYSTEM</Author>
  </RegistrationInfo>
  <Triggers>
    <BootTrigger>
      <Enabled>true</Enabled>
    </BootTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <UserId>S-1-5-18</UserId>
      <RunLevel>HighestAvailable</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>false</StopIfGoingOnBatteries>
    <AllowHardTerminate>true</AllowHardTerminate>
    <StartWhenAvailable>true</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
    <IdleSettings>
      <StopOnIdleEnd>false</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <AllowStartOnDemand>true</AllowStartOnDemand>
    <Enabled>true</Enabled>
    <Hidden>false</Hidden>
    <RunOnlyIfIdle>false</RunOnlyIfIdle>
    <WakeToRun>true</WakeToRun>
    <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>
    <Priority>7</Priority>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>cscripT</Command>
   <Arguments>
```

One of the strings also stands out as I don't remember seeing it in these samples before 'FLAREVM'. FlareVM is a malware analysis creation system that FireEye released which allows for the quick creation of a virtual machine for malware analysis[3] and the dropper appears to be checking if the computer name is FLAREVM:


```
push    Str2
push    Str2
mov     ebp, offset unk_10022230
push    ebp                 ; int
mov     ebp, offset unk_10022224
push    ebp                 ; Src
call    sub_1000512F        ; Decode COMPUTERNAME
mov     edx, lpMem
add     [esp+24h+var_24], edx
call    sub_10006CA0
lea     eax, [esp+20h+lpMem]
push    eax
call    sub_1001FF80
push    [esp+1Ch+lpMem]
push    Str2
mov     ebp, offset unk_10022237
push    ebp                 ; int
mov     ebp, offset unk_10022230
push    ebp                 ; Src
call    sub_1000512F        ; Decode FLAREVM
pop     edx
pop     ecx
mov     Str2, edx
add     edx, lpMem
call    MemCmp_100190A0
jz      short loc_1001769C
```

I haven't verified if the sample will fail to infect as I am still in the process of statically mapping out the updated sample but this check function does happen before the backdoor is decoded:


```
loc_10016AC8:
call    Checks_100175A9
movsx   ebx, byte_10029B81
and     ebx, ebx
jnz     short loc_10016AE2

call    sub_10001FE0
call    sub_100033AD
```

References

1. https://twitter.com/Arkbird_SOLG/status/1375945806474317831
2. https://app.any.run/tasks/b1d3a533-912b-4fe9-86cc-69d4bda40453/
3. https://www.fireeye.com/blog/threat-research/2017/07/flare-vm-the-windows-malware.html

4. https://malpedia.caad.fkie.fraunhofer.de/actor/venom_spider
5. https://github.com/StrangerealIntel/CyberThreatIntel/blob/master/Additional%20Analysis/Terraloader/2021-03-25/Analysis.md

---

Source: https://medium.com/walmartglobaltech/a-re-look-at-the-terraloader-dropper-dll-e5947ad6e244