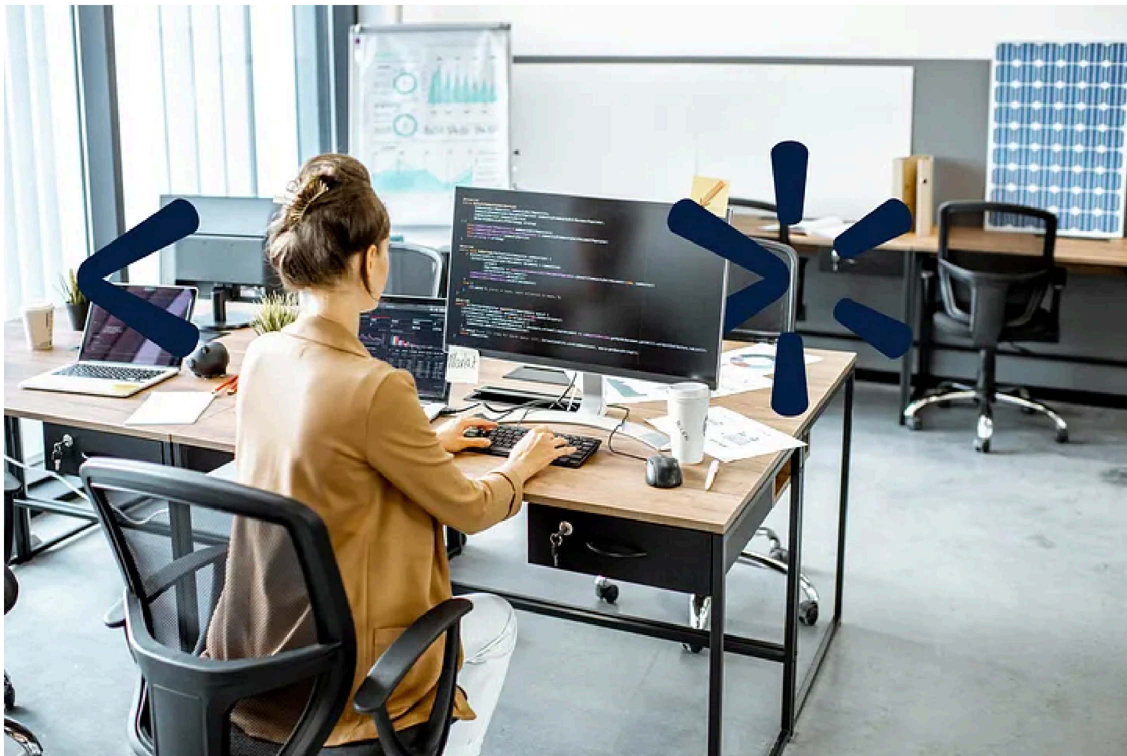
Decrypting BazarLoader strings with a Unicorn

By Jason Reaves

Published: 2021-07-30 · Archived: 2026-04-05 19:50:53 UTC

Press enter or click to view image in full size



BazarLoader[1,2,6] has been used by various teams over the past year primarily being leveraged for spam campaigns by teams associated with TrickBot[3]. While the initial malware changed the on objective TTPs (TrickBot Attack Team PlayBook) remain very similar to most of their infections that end with Anchor on high priority servers and ultimately ransomware infections[4,5,7].

Recently I've noticed on the Loader side that two versions have shown up using different obfuscations and in different campaigns. There are few different obfuscations being utilized by the various teams involved in Bazar but for the purpose of this report we will be focusing on the samples utilizing LLVM[8]. My aim is to show an interesting technique that I think goes under utilized in malware analysis where you can leverage a CPU emulator to decode out of various types of string encodings, I used this technique for many years to decode various portions of the H1N1[9] Loader and have also leveraged it for creating unpackers over the years such as with MAN1s old crypter[10].

Some of the unpacked samples we will be looking at refer to themselves as:

```
exeLoaderDll_LLVMO.dll
```

These samples store most of their relevant strings in an obfuscated manner where the data is manually loaded in and then ran through a fairly lengthy process of decoding the data.

```
0F 28 05 C0 A8 01 00     movaps  xmm0, cs:xmmword_14001F010
0F 29 00                 movaps  xmmword ptr [rax], xmm0
B9 FF AF 6B CA           mov     ecx, 0CA6BAFFFh
48 89 48 20              mov     [rax+20h], rcx
31 C0                    xor     eax, eax
48 8D 4C 24 74           lea     rcx, [rsp+108h+var_94]
88 41 FC                 mov     [rcx-4], al
C7 01 B3 72 D6 69        mov     dword ptr [rcx], 69D672B3h
C7 41 04 B7 76 D2 65     mov     dword ptr [rcx+4], 65D276B7h
C7 41 08 BB 7A DE 61     mov     dword ptr [rcx+8], 61DE7ABBh
C7 41 0C BF 7E DA 7D     mov     dword ptr [rcx+0Ch], 7DDA7EBFh
C7 41 10 A3 62 C1 7E     mov     dword ptr [rcx+10h], 7EC162A3h
C7 41 14 A7 66 C2 75     mov     dword ptr [rcx+14h], 75C266A7h
C7 41 18 AB 6A A5 1C     mov     dword ptr [rcx+18h], 1CA56AABh
C7 41 1C C0 03 A1 18     mov     dword ptr [rcx+1Ch], 18A103C0h
C7 41 20 C4 07 AD 14     mov     dword ptr [rcx+20h], 14AD07C4h
C7 41 24 F2 30 95 2D     mov     dword ptr [rcx+24h], 2D9530F2h
38 41 FC                 cmp     [rcx-4], al
0F 85 26 02 00 00        jnz     loc_1400049DA

41 B8 01 00 00 00        mov     r8d, 1
31 DB                    xor     ebx, ebx
41 BF FF FF FF FF        mov     r15d, 0FFFFFFFFh
41 BD 3B F4 84 3C        mov     r13d, 3C84F43Bh
41 B9 0E 4D 84 AF        mov     r9d, 0AF844D0Eh
```

Loading data

```
                         loc_1400047CE:
42 8B 6C 84 70           mov     ebp, [rsp+r8*4+108h+var_98]
89 E9                    mov     ecx, ebp
44 31 F9                 xor     ecx, r15d
89 C8                    mov     eax, ecx
25 47 24 42 14           and     eax, 14422447h
89 EE                    mov     esi, ebp
81 E6 00 90 20 41        and     esi, 41209000h
09 C6                    or      esi, eax
89 F0                    mov     eax, esi
35 00 00 20 40           xor     eax, 40200000h
81 F6 54 0F 03 80        xor     esi, 80030F54h
89 CF                    mov     edi, ecx
81 CF 44 94 02 01        or      edi, 1029444h
BA 54 9F 03 81           mov     edx, 81039F54h
21 D6                    and     esi, edx
```

Start of decode loop

Investigating more instances of this process in the same sample shows variations meaning it was dynamically generated whether using macros or a lower level obfuscator, the TrickBot group has historically utilized both ADVobfuscator[11] and LLVM[8].

For decoding the strings with an emulator[12] we will need to capture the block of data that loads the bytes and also the loop that decodes it, luckily for obfuscators like this there are normally patterns we can signature on for the samples:

```
import sys
import re
import binascii
import struct
from unicorn import *
from unicorn.x86_const import *STACK=0x90000
code_base = 0x10000000
mu = Uc(UC_ARCH_X86,UC_MODE_64)data = open(sys.argv[1], 'rb').read()
test = re.findall(r'''488d.{3,20}c70.+0f......ffff''',binascii.hexlify(data))
```

In this code block we are doing some initial setup for unicorn[12] and then finding our block of code, because of how python regex is being used here we will get the first block in the file all the way through the last block. This means we will need to break up and parse out the individual blocks, I prefer this method because it lets me take control of the process to a degree.

```
temp = test[0]
temp = ['488d'+x for x in temp.split('488d')]
tempp = []
for x in temp:
 xx = x.split('feffff')
 if 'fdffff' in xx[0]:
  xx = x.split('fdffff')
  tempp.append(xx[0]+'fdffff')
 else:
  tempp.append(xx[0]+'feffff')
temp = tempp
```

So we break up each block by the start and end while accounting for a variation that I noticed in some of the samples for the ending bytes. Up next we will finish setting up our emulator and then loop through and emulate each block of code:

```
mu.mem_map(code_base, 0x100000)mu.mem_map(STACK, 4096*10)
for i in range(len(temp)):
 try:
  blob = binascii.unhexlify(temp[i])
 except:
  blob = binascii.unhexlify(temp[i][1:])
 mu.mem_write(code_base, '\x00'*0x100000)
```

```
mu.mem_write(STACK, '\x00'*(4096*10))

 mu.mem_write(code_base,blob)
mu.reg_write(UC_X86_REG_ESP,STACK+4096)
try:
 mu.emu_start(code_base, code_base+len(blob), timeout=10000)
except:
 pass
```

After emulation we will read in the entire memory we had allocated for the stack and then print out any strings found by stripping all NULL bytes:

```
a = mu.mem_read(STACK,4096*10)
a = a[len(blob):].split('\x00')
a = filter(lambda x: x != '', a)
a = map(str,a)
print(str(''.join(a)))
```

An example gives us a healthy chunk of data:

```
# python str_decode.py 9d76e72fb45bb059b64c58d10da43cbac1487f8b396d705eae0a427974587171.bin  |string
Mozilla/5.0
ABCDEFGHIJKLMNOPQRTSUVWXYZ0123456789
%s.%d.%d.%d%s.%s.%d.%d.%d%s.%s
%%s.%d.%d.%d%s.%s.%d.%d.%d%s.%s
 Avast.exe
c34.212.193.150 35.166.147.40
rareanimalsofcanada.bazar wildwinternature.bazar coldmountainsanimals.bazar
Software\%s
cmd /c ping 8.8.7.7 -n 2 &
8Y3y
start %s %s
GGNY
 yyyy-MM-dd
SHA384
HashDigestLength8
 ECDSA_P384
kernel32.dll
@advapi32.dll
xuser32.dll
ws2_g
+ntdll.dll
shell32.dll
crypt32.dll
shlwapi.dll
```

```
owinhttp.dll
netapi32.dll
bcrypt.dll
userenv.dll
```

Running on a set of files from VirusTotal we can quickly churn out some C2 lists.

Get Jason Reaves’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Samples:

```
0c2e254376127f76d44fc9276000697e45a2977fca4384705e84994ab63fdc37  90d0c4995ce53077cd2fbc00a248f02df1
0d53ed1eca1a3d28e0227055f66f6d2de1b2606e30f10982967509df4e478ee1  9296e1fef0356eab2956aac2d010dac587
1c27d4dc6fef72e096b06662908d7c5b225cecdbfc66c8f19db78b76008fac63
270890cfa6621fa3b5c6edcdd2bb15760b97abd43245d6673eee9dca23c77d40
2ea153ff7675c15adcda2bff88958be2004f9d32f6d67d9fabd3c872eeb07505
2eec5366c21fc1bc9c11c2afbf66368fc704175231ab63659b3e8b839e5c9e71
37aae88b9a3f942952c258d611c2c629116fcc077079e3698590c3f8aab3e684
37e587e6b801e926dc31da093c55f1f834edcb8c1971c40869a8054580e39e42  9d76e72fb45bb059b64c58d10da43cbac1
447b4c867b7147afe178d73adf8113fc33f6399f03707e4308efa36e0859bf86  9f6ae735999f98738022b1784d1b46975a
47eb57d467c4330269a5238a53dd399c5183b338a8bbead88bb8b88b4396a80c  ae6e6dd4f2aa22ccc395ade0ae713000af
5791ef7d6916f8c14d3261a9c3d9b68b30e208e2ddd74d0dae1ad0a476504e2a  c0a087a520fdfb5f1e235618b3a5101969
664e8512cd3ce3552f33878e26800184e0cff8ee54c75bfa93f19ad97615bb56
68b4f6fde1a2d1024f4028d22d12daeaf3f4ae4ffb46cc07cf11cf6a2cb35e90  d5df7e82b5ff898d49f3f779f20644916
69f897a4ccf41cdf3f0c7903fc740b6914707d3286a5b5c8ca1ff90487b1c4ef  e06473cad41789dddc88aa58b2f143302
87ad0b1bd7a18ff2aa975991cd15725b4ddfa0d0ef972cbe2f57a789582aa675  f18c2a8922bbe7b8f12980a46cc3548e9a
8a0fbcde56a9a817c10b0fe5ae281f75385c2a28ca271d736484e689c104e96c  f29253139dab900b763ef436931213387d
```

IPs, Domains and URIs:

```
18.188.232.155
18.191.220.165
18.222.240.99
194.5.249.30
3.134.106.170
34.209.40.84
34.212.193.150
35.166.147.40
45.142.158.252
54.184.178.68
54.184.52.204
54.190.171.88
```

```
acghilbdihio.bazar
achiikbdjiin.bazar
adehjkbeghjn.bazar
adfhikbehhin.bazar
adggklbeigko.bazar
afegkmbgggkp.bazar
bceikkccgikn.bazar
bchgjlcdjgjo.bazar
begiklceiiko.bazar
bffhklcghhko.bazar
cegiikdeiiin.bazar
coastalbrezecarwash.com
coldmountainsanimals.bazar
deehimeeghip.bazar
rareanimalsofcanada.bazar
wildwinternature.bazar
/thirst/honor/commerce
/no/link/1
```

References

1: https://www.bleepingcomputer.com/news/security/bazarbackdoor-trickbot-gang-s-new-stealthy-network-hacking-malware/

2: https://blog.fox-it.com/2020/06/02/in-depth-analysis-of-the-new-team9-malware-family/

3: https://cybersecurity.att.com/blogs/labs-research/trickbot-bazarloader-in-depth

4: https://thedfirreport.com/2021/03/08/bazar-drops-the-anchor/

5: https://thedfirreport.com/2020/10/18/ryuk-in-5-hours/

6: https://malpedia.caad.fkie.fraunhofer.de/details/win.bazarbackdoor

7: https://labs.sentinelone.com/inside-a-trickbot-cobaltstrike-attack-server/

8: https://github.com/obfuscator-llvm/obfuscator

9: https://malpedia.caad.fkie.fraunhofer.de/details/win.h1n1

10: https://vixra.org/pdf/1902.0257v1.pdf

11: https://github.com/andrivet/ADVobfuscator

12: https://www.unicorn-engine.org/docs/

Source: https://medium.com/walmartglobaltech/decrypting-bazarloader-strings-with-a-unicorn-15d2585272a9