Decrypting BazarLoader strings with a Unicorn

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

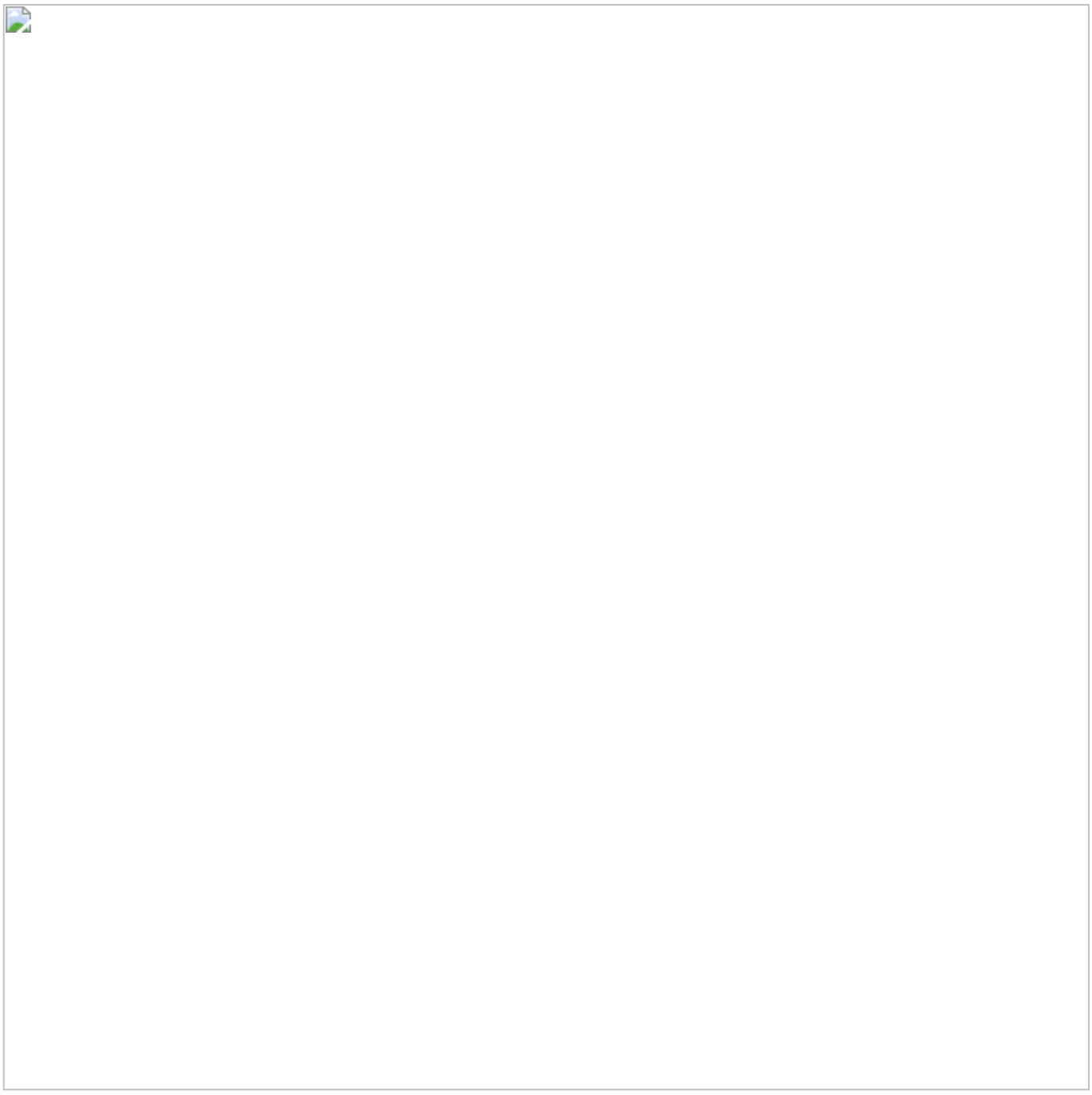
Jason Reaves July 30, 2021

Jason Reaves

Jul 30, 2021

.

4 min read



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].

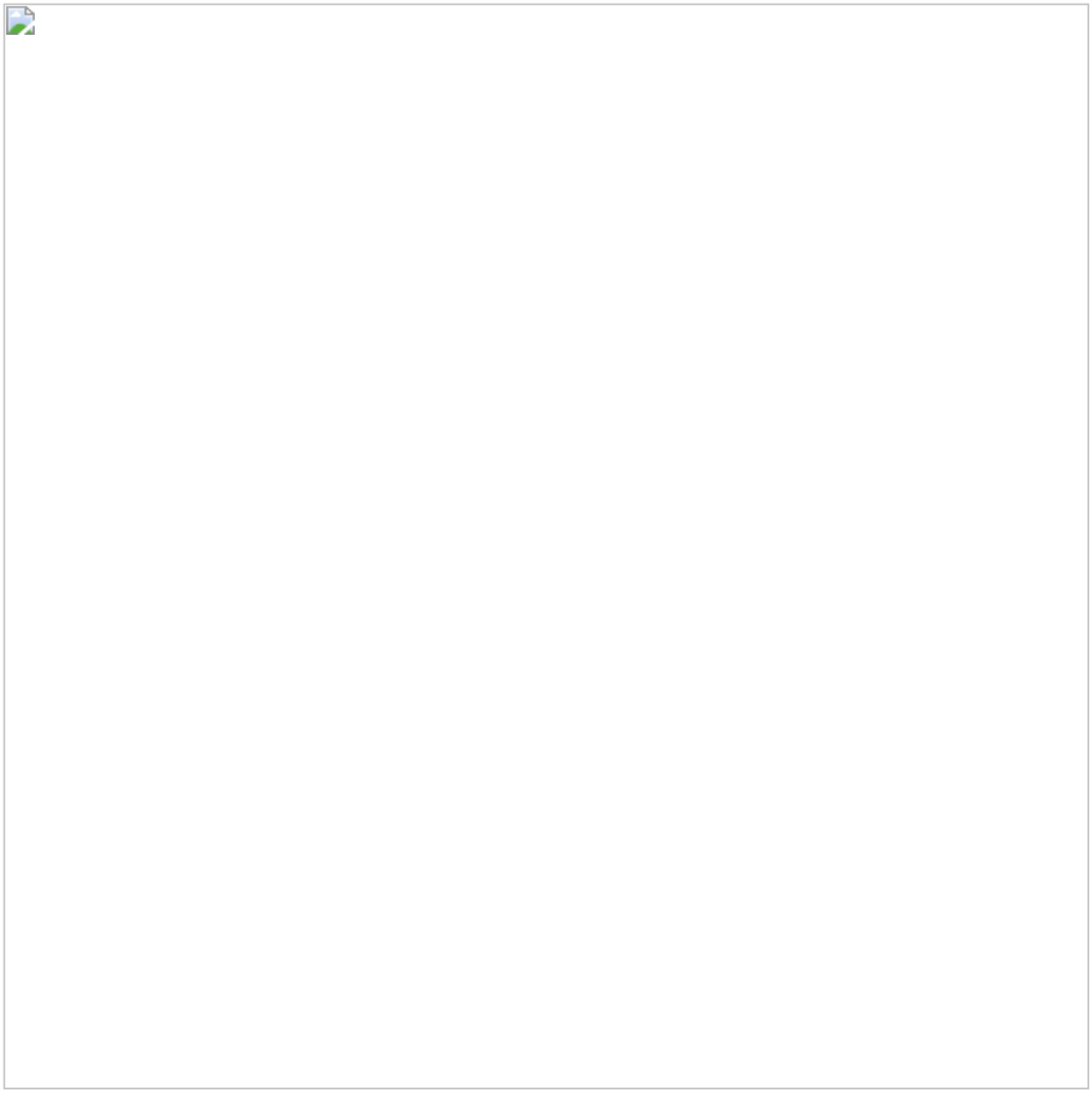
Recently I've noticed on the Loader side that two versions have shown up using different obfuscations and in different campaigns. There are few different obfuscations being utilized by the various teams involved in Bazar but for the purpose of this report we will be focusing on the samples utilizing 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].

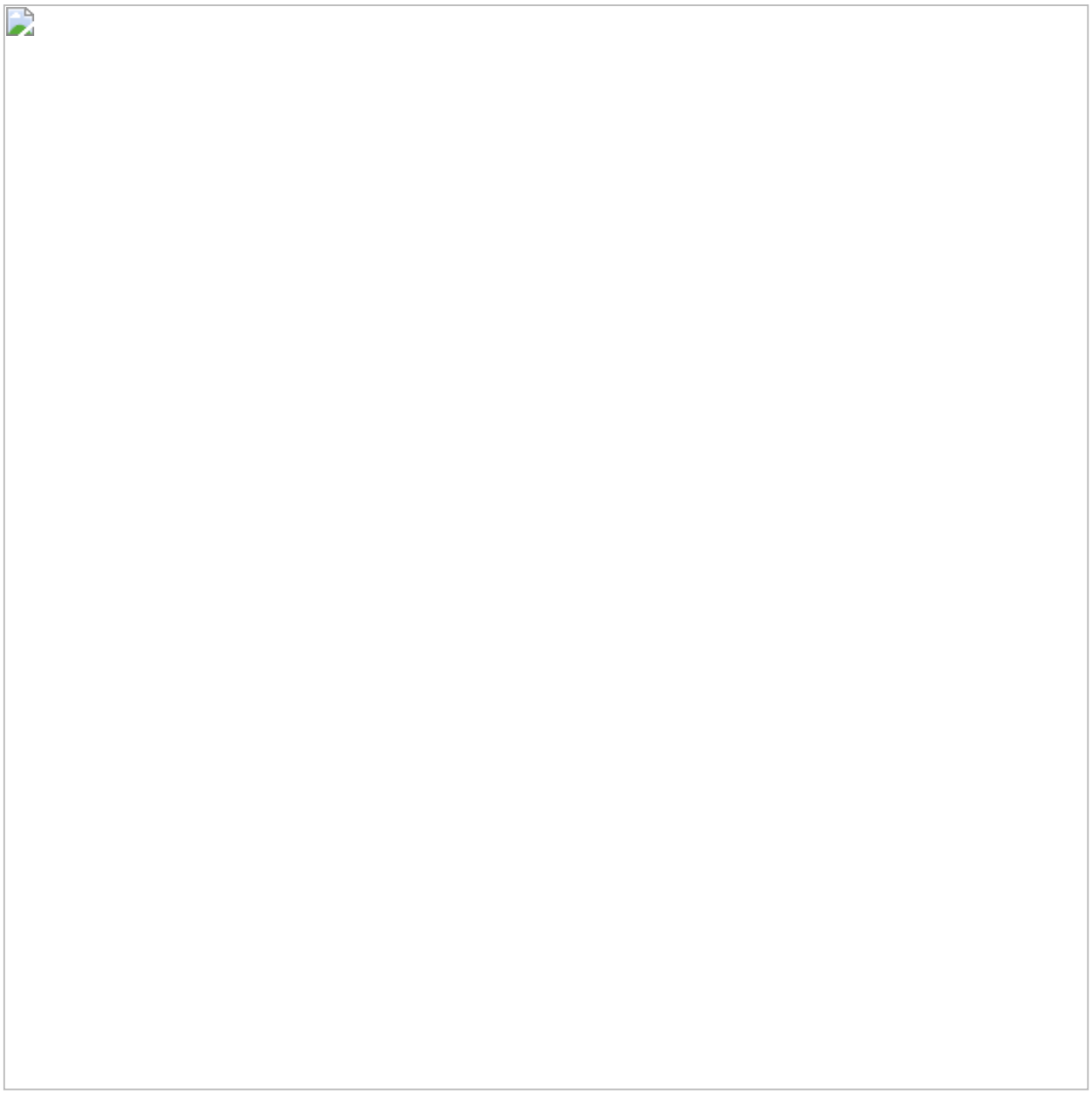
Some of the unpacked samples we will be looking at refer to themselves as:

```
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.



Loading data



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 sysimport reimport binasciiimport structfrom unicorn import *from
unicorn.x86_const import *STACK=0x90000code_base = 0x10000000mu =
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
|stringsMozilla/5.0ABCDEFGHIJKLMNOPQRTSUVWXYZ0123456789%s.%d.%d.%d%s.%s.%d.%d.%d%s.%s%%s
 Avast.exec34.212.193.150 35.166.147.40rareanimalsofcanada.bazar wildwinternature.bazar
coldmountainsanimals.bazarSoftware\%scmd /c ping 8.8.7.7 -n 2 & 8Y3ystart %s %sGGNY
yyyy-MM-ddSHA384HashDigestLength8
ECDSA_P384kernel32.dll.dllxuser32.dllws2_g+ntdll.dllshell32.dllcrypt32.dllshlwapi.dllowi
```

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

Samples:

```
0c2e254376127f76d44fc9276000697e45a2977fca4384705e84994ab63fdc37
90d0c4995ce53077cd2fbc00a248f02df108b42b4df1ba84b89ea014fae4ea010d53ed1eca1a3d28e0227055

9296e1fef0356eab2956aac2d010dac587d55ae2de7f520deb97d1dfcc4e4a9a1c27d4dc6fef72e096b06662
  270890cfa6621fa3b5c6edcdd2bb15760b97abd43245d6673eee9dca23c77d40
2ea153ff7675c15adcda2bff88958be2004f9d32f6d67d9fabd3c872eeb075052eec5366c21fc1bc9c11c2af
  37aae88b9a3f942952c258d611c2c629116fcc077079e3698590c3f8aab3e684
37e587e6b801e926dc31da093c55f1f834edcb8c1971c40869a8054580e39e42
9d76e72fb45bb059b64c58d10da43cbac1487f8b396d705eae0a427974587171447b4c867b7147afe178d73a

9f6ae735999f98738022b1784d1b46975ae16069c260656646fbcfeaeef35a0647eb57d467c4330269a5238a

ae6e6dd4f2aa22ccc395ade0ae713000af9d3dc189651e054b46540647ec891c5791ef7d6916f8c14d3261a9

c0a087a520fdfb5f1e235618b3a5101969c1de85b498bc4670372c02756efd55664e8512cd3ce3552f33878e
  68b4f6fde1a2d1024f4028d22d12daeaf3f4ae4ffb46cc07cf11cf6a2cb35e90
d5df7e82b5ff898d49f3f779f2064491654ae3d50129aa0bd48a88cd43c4211369f897a4ccf41cdf3f0c7903

e06473cad41789dddc88aa58b2f1433023637c468a88bbe364db50c1e451374487ad0b1bd7a18ff2aa975991

f18c2a8922bbe7b8f12980a46cc3548e9a0903a7294206eeb2d01f7923cdb8eb8a0fbcde56a9a817c10b0fe5
  f29253139dab900b763ef436931213387dc92e860b9d3abb7dcd46040ac28a0e
```

IPs, Domains and URIs:

```
18.188.232.15518.191.220.16518.222.240.99194.5.249.303.134.106.17034.209.40.8434.212.193
```

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/