MetaStealer: String Decryption and DGA overview

By Jason Reaves

Published: 2023-05-09 · Archived: 2026-04-10 02:35:35 UTC

5 min read

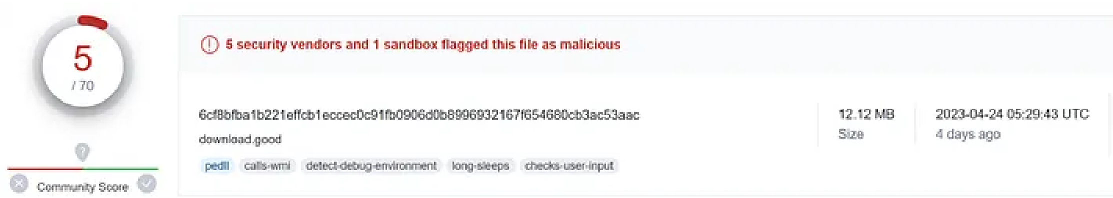
May 9, 2023

By: Jonathan McCay, Joshua Platt and Jason Reaves

Unit42[1] recently tweeted about a campaign starting with a malicious email link that downloads a OneNote file used to drop and execute MetaStealer. While investigating the MetaStealer sample[2], we noticed it attempts to connect to multiple domains that seemed to be randomly named. After landing on the C2 routine, instead of decrypting a static list of servers, the sample used a domain generation algorithm[3], (DGA) to derive the list. Predominantly used as a fall back mechanism, a domain from the DGA can be registered to communicate when the primary C2s are not available. The use of a DGA as a primary method of contacting the threat actor's infrastructure is interesting.

Sample:

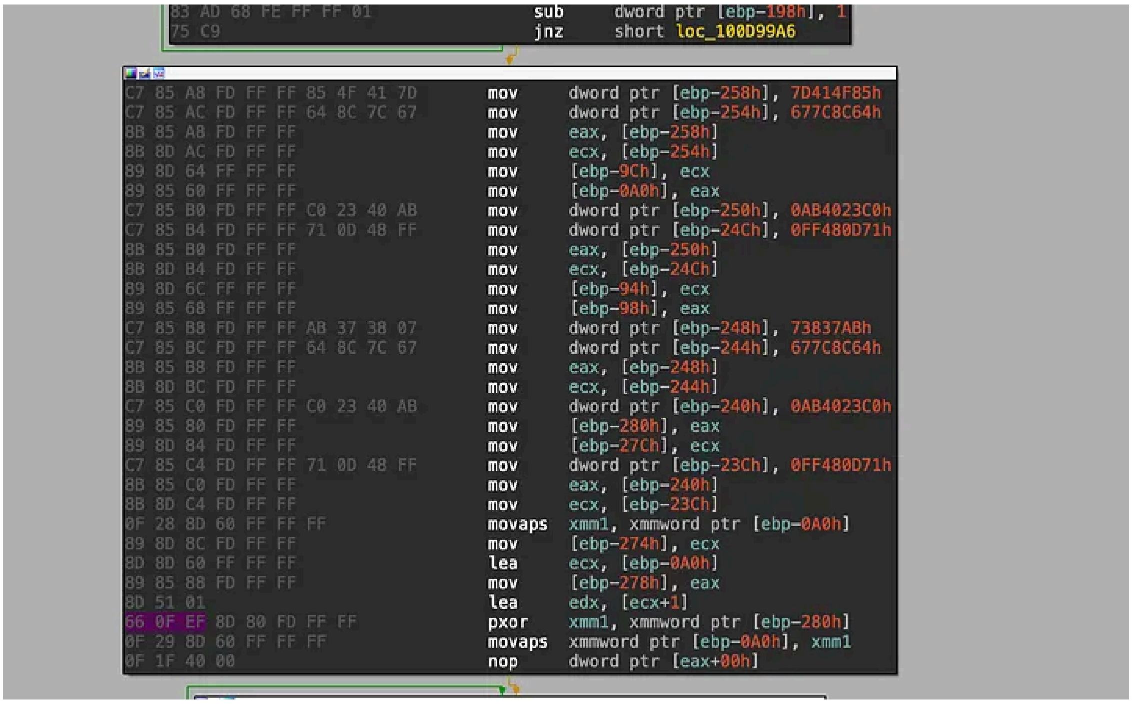
Press enter or click to view image in full size



String Decryption:

On top of the MetaStealer binary being highly obfuscated, the author inserted garbage code while also encoding the important onboard strings. The encoding used has been previously mentioned by NCC[4], but a lack of listing of the decoded strings in existing research was noticed. The strings are organized as DWORD values and pushed onto the stack along with the XOR key:

Press enter or click to view image in full size

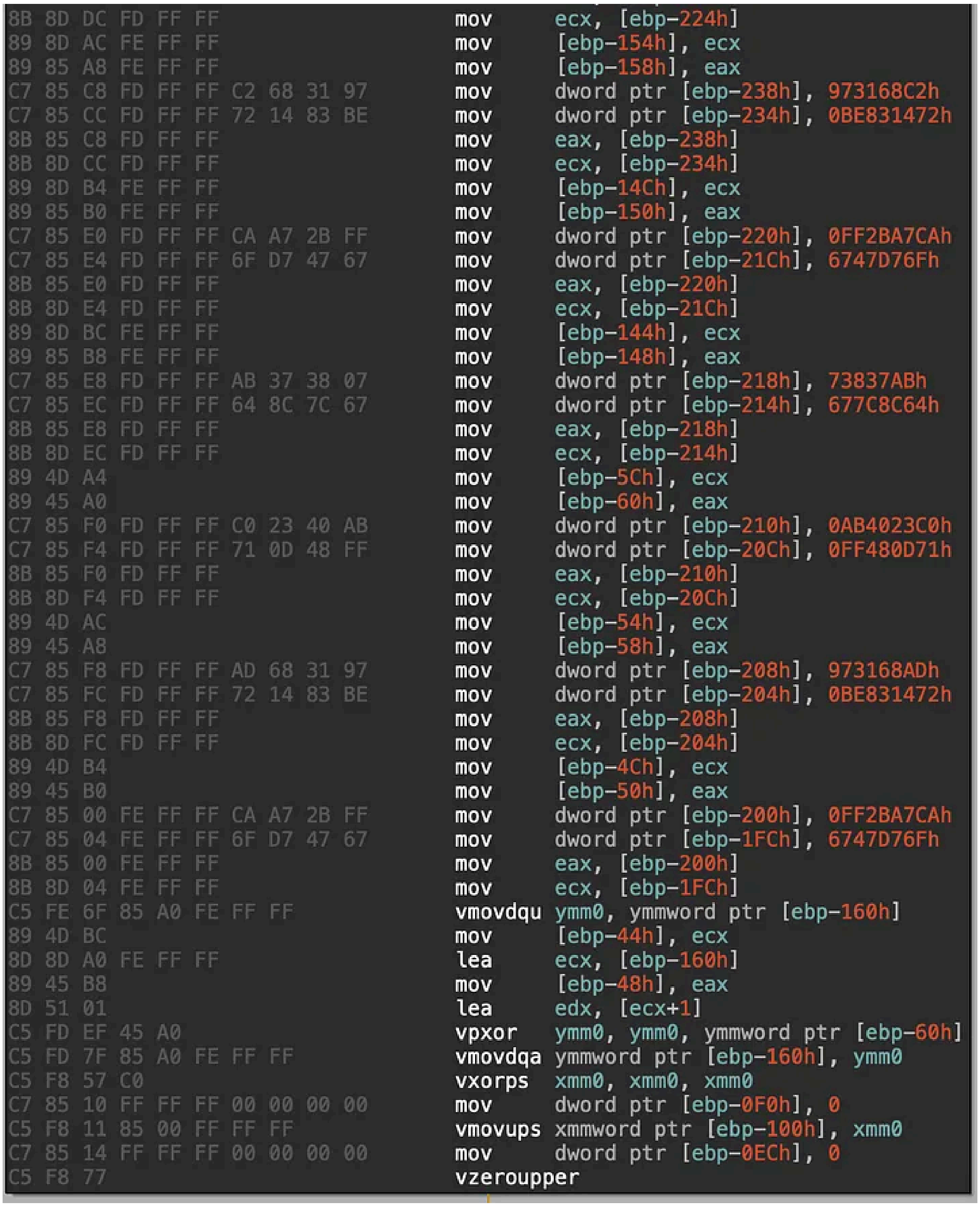


The PXOR instruction allows for using 64 bit(MMX) or 128 bit(XMM) length operands. In this case, you can see that most of the DWORD values are the same. This is because the encoded strings are NULL padded to make the length fit. Ultimately, in the screen above, the data is just XOR encoded:

```
>>> a = struct.pack('<IIII', 0x7D414F85, 0x677C8C64, 0x0AB4023C0, 0x0FF480D71)
>>> b = struct.pack('<IIII', 0x73837AB, 0x677C8C64, 0x0AB4023C0, 0x0FF480D71)
>>> a = bytearray(a)
>>> b = bytearray(b)
>>> for i in range(len(a)):
...   a[i] ^= b[i]
...
>>> a
bytearray(b'.xyz\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
```

Attempting to decode all or the majority of the strings in an automatic manner would require finding the relevant blocks of code. After some investigation, it seems longer strings use the VPXOR instruction, which can leverage 128 bit(XMM) or 256 bit(YMM) length operands.

Press enter or click to view image in full size



```
8B 8D DC FD FF FF                  mov     ecx, [ebp-224h]
89 8D AC FE FF FF                  mov     [ebp-154h], ecx
89 85 A8 FE FF FF                  mov     [ebp-158h], eax
C7 85 C8 FD FF FF C2 68 31 97      mov     dword ptr [ebp-238h], 973168C2h
C7 85 CC FD FF FF 72 14 83 BE      mov     dword ptr [ebp-234h], 0BE831472h
8B 85 C8 FD FF FF                  mov     eax, [ebp-238h]
8B 8D CC FD FF FF                  mov     ecx, [ebp-234h]
89 8D B4 FE FF FF                  mov     [ebp-14Ch], ecx
89 85 B0 FE FF FF                  mov     [ebp-150h], eax
C7 85 E0 FD FF FF CA A7 2B FF      mov     dword ptr [ebp-220h], 0FF2BA7CAh
C7 85 E4 FD FF FF 6F D7 47 67      mov     dword ptr [ebp-21Ch], 6747D76Fh
8B 85 E0 FD FF FF                  mov     eax, [ebp-220h]
8B 8D E4 FD FF FF                  mov     ecx, [ebp-21Ch]
89 8D BC FE FF FF                  mov     [ebp-144h], ecx
89 85 B8 FE FF FF                  mov     [ebp-148h], eax
C7 85 E8 FD FF FF AB 37 38 07      mov     dword ptr [ebp-218h], 73837ABh
C7 85 EC FD FF FF 64 8C 7C 67      mov     dword ptr [ebp-214h], 677C8C64h
8B 85 E8 FD FF FF                  mov     eax, [ebp-218h]
8B 8D EC FD FF FF                  mov     ecx, [ebp-214h]
89 4D A4                           mov     [ebp-5Ch], ecx
89 45 A0                           mov     [ebp-60h], eax
C7 85 F0 FD FF FF C0 23 40 AB      mov     dword ptr [ebp-210h], 0AB4023C0h
C7 85 F4 FD FF FF 71 0D 48 FF      mov     dword ptr [ebp-20Ch], 0FF480D71h
8B 85 F0 FD FF FF                  mov     eax, [ebp-210h]
8B 8D F4 FD FF FF                  mov     ecx, [ebp-20Ch]
89 4D AC                           mov     [ebp-54h], ecx
89 45 A8                           mov     [ebp-58h], eax
C7 85 F8 FD FF FF AD 68 31 97      mov     dword ptr [ebp-208h], 973168ADh
C7 85 FC FD FF FF 72 14 83 BE      mov     dword ptr [ebp-204h], 0BE831472h
8B 85 F8 FD FF FF                  mov     eax, [ebp-208h]
8B 8D FC FD FF FF                  mov     ecx, [ebp-204h]
89 4D B4                           mov     [ebp-4Ch], ecx
89 45 B0                           mov     [ebp-50h], eax
C7 85 00 FE FF FF CA A7 2B FF      mov     dword ptr [ebp-200h], 0FF2BA7CAh
C7 85 04 FE FF FF 6F D7 47 67      mov     dword ptr [ebp-1FCh], 6747D76Fh
8B 85 00 FE FF FF                  mov     eax, [ebp-200h]
8B 8D 04 FE FF FF                  mov     ecx, [ebp-1FCh]
C5 FE 6F 85 A0 FE FF FF            vmovdqu ymm0, ymmword ptr [ebp-160h]
89 4D BC                           mov     [ebp-44h], ecx
8D 8D A0 FE FF FF                  lea     ecx, [ebp-160h]
89 45 B8                           mov     [ebp-48h], eax
8D 51 01                           lea     edx, [ecx+1]
C5 FD EF 45 A0                     vpxor   ymm0, ymm0, ymmword ptr [ebp-60h]
C5 FD 7F 85 A0 FE FF FF            vmovdqa ymmword ptr [ebp-160h], ymm0
C5 F8 57 C0                        vxorps  xmm0, xmm0, xmm0
C7 85 10 FF FF FF 00 00 00 00      mov     dword ptr [ebp-0F0h], 0
C5 F8 11 85 00 FF FF FF            vmovups xmmword ptr [ebp-100h], xmm0
C7 85 14 FF FF FF 00 00 00 00      mov     dword ptr [ebp-0ECh], 0
C5 F8 77                           vzeroupper
```

A quick and dirty way to decode strings is to find all the PXOR and VPXOR instructions and then walk backwards to find all the DWORD stack loads to get most of the strings decoded.

```
rule_source = '''
rule meta_s
{
    meta:
        author = "sysopfb"
    strings:
        $snippet1 = {66 0? ef}
        $snippet2 = {c5 ?? ef}
```

```
    condition:
        ($snippet1 or $snippet2)
}
'''
def yara_scan(raw_data, rule_name):
    addresses = {}
    yara_rules = yara.compile(source=rule_source)
    matches = yara_rules.match(data=raw_data)
    for match in matches:
        if match.rule == 'meta_s':
            for item in match.strings:
                if item.identifier == rule_name:
                    addresses[item.identifier] = item.instances
    return addresses

data = open(sys.argv[1], 'rb').read()
ret = []
out = None

length_off = False
snippet = yara_scan(data, '$snippet1')
prev_offset = 0
for val in snippet['$snippet1']:
 offset = val.offset
 test = data[prev_offset:offset]
 vals = re.findall(b'''\xc7\x85..\xff\xff....''',test)
 if vals != []:
  if len(vals) > 8:
   temp = vals[-8:]
  else:
   temp = vals
  try:
   xdata = temp[0][-4:]
   xdata += temp[1][-4:]
   xdata += temp[2][-4:]
   xdata += temp[3][-4:]
   xkey = temp[4][-4:]
   xkey += temp[5][-4:]
   xkey += temp[6][-4:]
   xkey += temp[7][-4:]
   xdata = bytearray(xdata)
   xkey = bytearray(xkey)
   for i in range(len(xdata)):
    xdata[i] ^= xkey[i]
   print(b''.join(xdata.split(b'\x00')))
  except:
   pass
 prev_offset = offset
```

Then repeating a similar approach for PVXOR but accounting for larger assortment of strings, as previously mentioned this will only get a majority of the strings decoded but it is enough to determine that this stealer has much functionality including references to starting socks, backconnect, punching holes in the firewall and detonating shellcode:

Decrypted Strings:

```
sys
chrome
firefox
edge
notepad
ip
password
cmd
input
FG Started
shellcode
mode
Result:
bc_addr
port
socks started
:1775
uuid
/api/client/new
ffox
dir=in
version
.xyz
ok
os_crypt
encrypted_key
RtlGetVersion
Windows 10
Windows Vista
Windows 7
Windows 8
Windows 8.1
Pro
action
status
passwd
loader_id
files
dur
```

```
script
FALSE
type
task_result
/tasks/collect
ROOT
name
value
host_key
is_httponly
httponly
path
is_secure
secure
expires_utc
encrypted_value
domain
expirationDate
host
isHttpOnly
isSecure
expiry
origin_url
password_value
url
ntdll
File-Type
cmd.exe
microsoft\\windows
/c "echo start "" "
Failed to create task definition
action=allow program="
(\\d{1,3})\\.(\\d{1,3})
\\Default\\Login Data
\\Default\\Web Data
```

A more exhaustive method for harvesting stack loaded and then decoded strings would be using a CPU emulator similar to how we showed against BazaLoader[5] previously.

Domain Generation Algorithm:

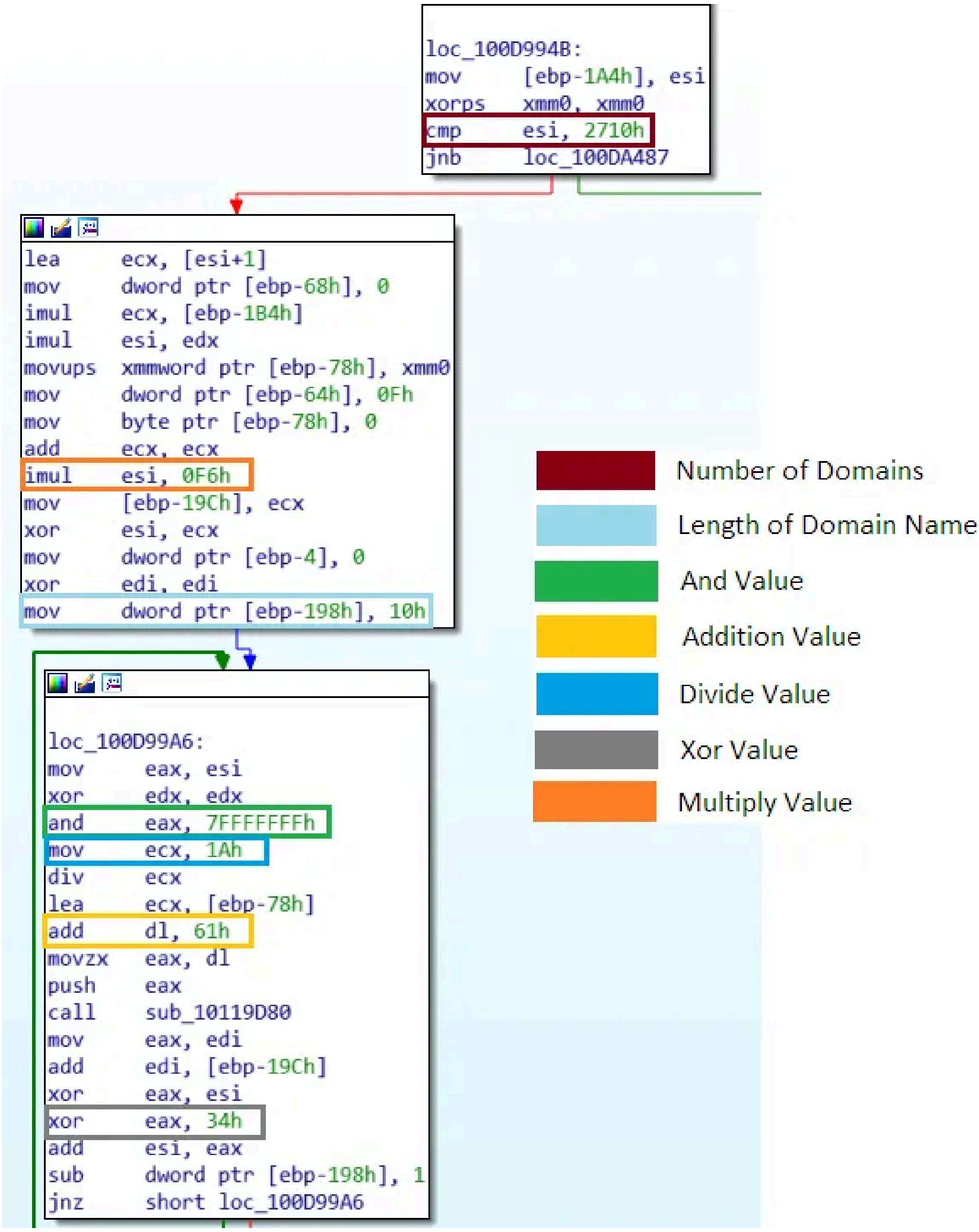
Additional domains the sample attempts to communicate with that appeared to be DGA.

uoyqwyekeyaiuiks.xyz	0 / 86	-
uoyyikuaukmcmmoe.xyz	0 / 86	-
wgcuuiqqasmoyyoy.xyz	0 / 86	-
wgcuwcgociewewoo.xyz	1 / 87	2023-03-14
wggceuugsyeeoqwk.xyz	0 / 86	-
wgiaeqkooyegkooc.xyz	0 / 86	-
wgkigqasaaysewkk.xyz	0 / 86	-
wgkycqwcowwwsgow.xyz	0 / 86	-

IDA: Pushing seed, (0x1234) before calling DGA

```
lea     eax, [ebp-210h]
push    1234h
push    eax
mov     [ebp-278h], edi
```

IDA: DGA Routine

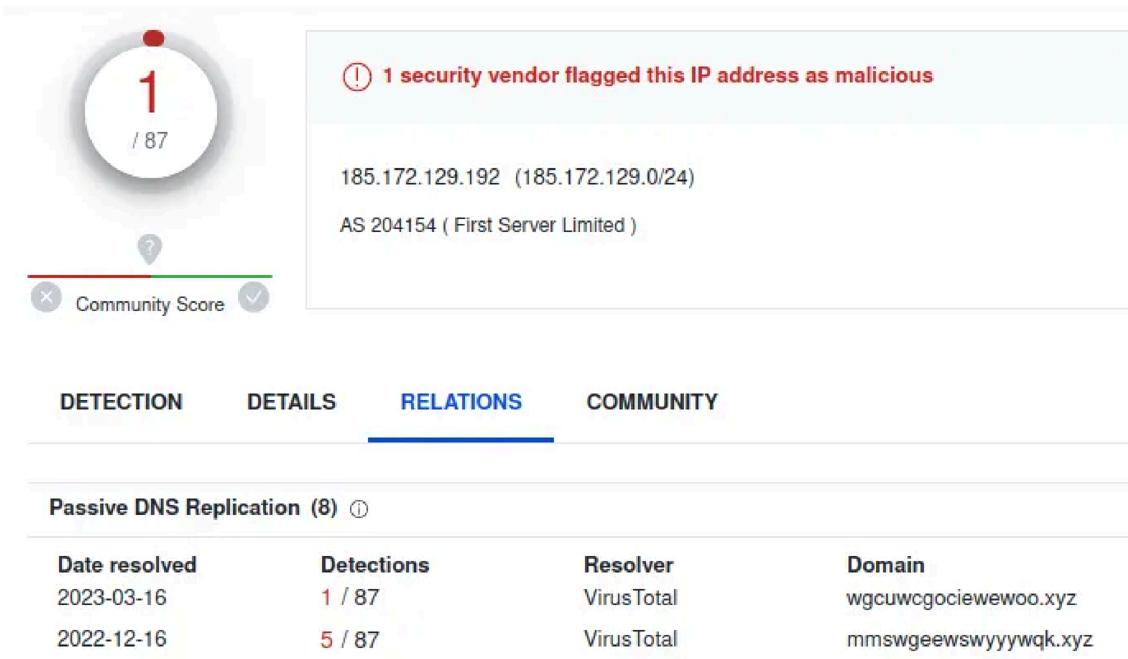


Converted to Python:

Press enter or click to view image in full size

```
num_of_domains = 0x2710
len_of_domain = 0x10
and_value = 0x7FFFFFFF
add_value = 0x61
div_value = 0x1A
xor_value = 0x34
mul_value = 0xF6

seed = 0x1234
seedling_1 = seed
seedling_2 = seed

for i in range(num_of_domains):
    domain = b''
    sapling_1 = 2 * seedling_2 * (i + 1)
    sapling_2 = sapling_1 ^ mul_value * seedling_1 * i
    xor_val = 0

    for i in range(len_of_domain):
        domain += bytes([((sapling_2 & and_value) % div_value) + add_value])
        xor_value_2 = xor_val
        xor_val += sapling_1
        sapling_2 += sapling_2 ^ xor_value_2 ^ xor_value

    print(domain)
```

Now that the host names have been generated, the url can be built using the tld and uri from the decrypted strings. An attempt to connect to each host will be made until a response is received. At the time of writing this, only one of the derived hostnames, (wgcuwcgociewewoo.xyz) would resolve. The ip it resolves to, (pictured below) also contains another hostname that used to be active, (mmswgeewswyyywqk.xyz). Both of these domains are in the list derived by the DGA when using the seed value 0x1234. Going back further we can find additional seed values being used at different time periods; 0xabc8, 0x9b2f, 0x7b2f, 0xc17a, 0x2f73 and 0x4b9a.



When KELA[6] first observed the threat actor "_META_" offering a new stealer, it was marketed as having the same functionality and panel as RedLine[7] Stealer. After seeing references to backconnect, socks, and loader id's in the decrypted strings, we can see that the improvements made to this tool now offer more than just credential theft.

IOCs

Endpoint:

```
powershell -inputformat none -outputformat none –NonInteractive -Command Add-MpPreference -Exclusion
AppData\Local\Microsoft\Windows\hyper-v.exe
hyper-v.ver
```

Network:

```
mmswgeewswyyywqk.xyz
wgcuwcgociewewoo.xyz
yiogqkksoyysqiky.xyz
ikuasuggwiewymsi.xyz
uosqysascuwmqgyk.xyz
uiouaqcqqcgueweg.xyz
uawqgawkguwiqeyk.xyz
kaewquswkswcmsim.xyz
aaycciywcgaqwkky.xyz
mkgcsmogqewauaiw.xyz
185.172.129.192
185.203.116.71
167.88.12.112

[a-z]{16}.xyz:1775/api/client/new
[a-z]{16}.xyz:1775/api/client/verify
[a-z]{16}.xyz:1775/api/client_hello
[a-z]{16}.xyz:1775/tasks/get_worker
[a-z]{16}.xyz:1775/tasks/collect
[a-z]{16}.xyz:1775/avast_update
```

Full DGA dump for every currently known seed can be found at:
https://github.com/sysopfb/open_mal_analysis_notes/tree/master/metastealer_dga

References

1: https://twitter.com/Unit42_Intel/status/1646940355936256000

2: https://www.virustotal.com/gui/file/6cf8bfba1b221effcb1eccec0c91fb0906d0b8996932167f654680cb3ac53aac

3: https://en.wikipedia.org/wiki/Domain_generation_algorithm

4: https://research.nccgroup.com/2022/05/20/metastealer-filling-the-racoon-void/

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

6: https://www.bleepingcomputer.com/news/security/new-blackguard-password-stealing-malware-sold-on-hacker-forums/

7: https://www.proofpoint.com/us/blog/threat-insight/new-redline-stealer-distributed-using-coronavirus-themed-email-campaign

---