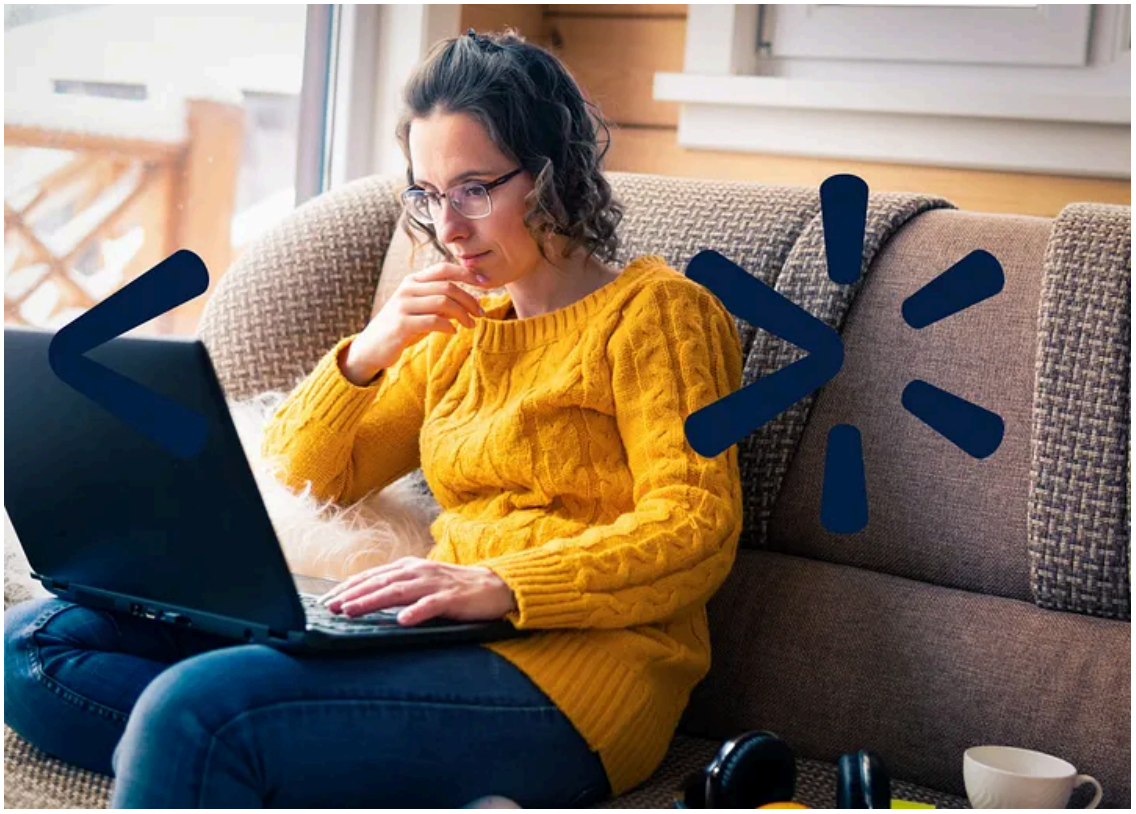
CobaltStrike UUID stager

By Jason Reaves

Published: 2022-03-28 · Archived: 2026-04-05 19:09:45 UTC

By: Jason Reaves

Press enter or click to view image in full size



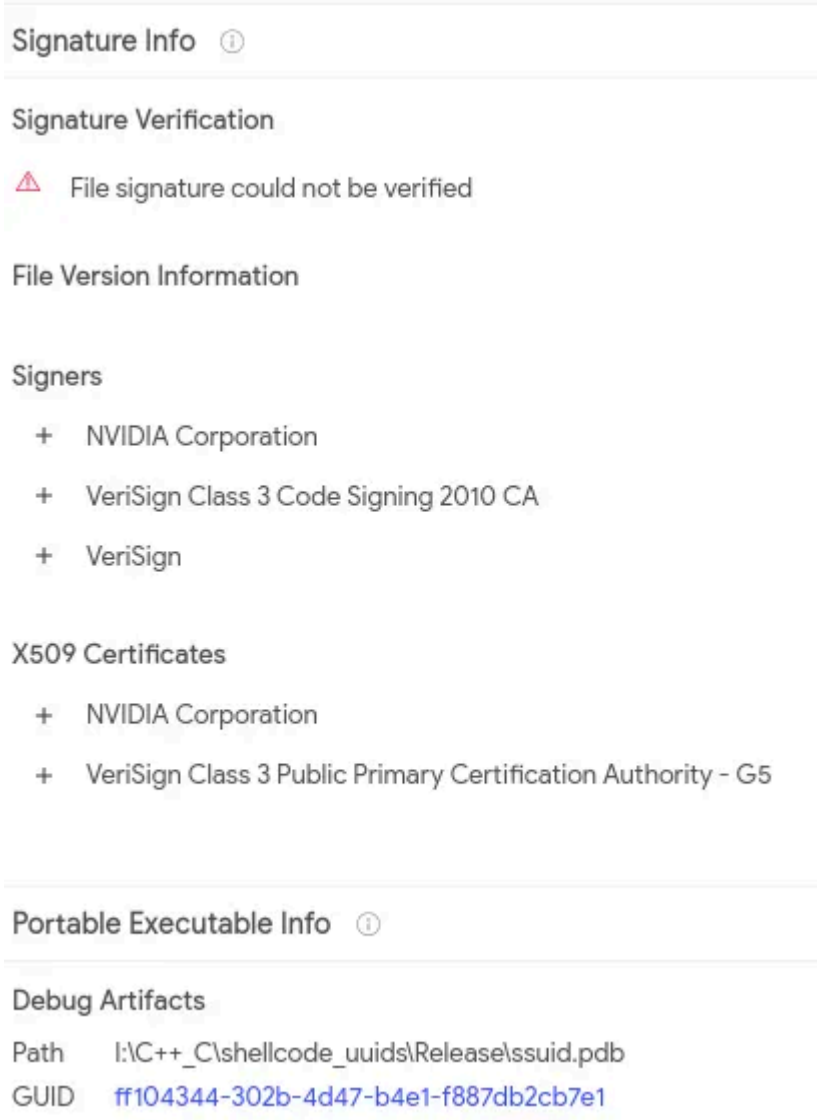
Lots of interesting stagers and loaders exist for CobaltStrike, we've talked about a few them written in various languages[1,2,3]. We have been tracking one for over a year now which hexlifies the stager code on board, recently we saw a hit for a YARA rule we use to track this stager:

```
rule cs_hexlified_stager_sc
{
strings:
$a1 = "d2648b52308b" nocase
condition:
all of them
}
```

Sample:

```
488d4cbe017c493c66a769ccb203b40cbca3396c654bfff72c1aebeb41f23297
```

However our system for decoding out the shellcode failed so we took a closer look and noticed a few interesting things, one the hexlified shellcode was stored as UUIDs[4] and two the sample was signed by NVIDIAs cert which was recently leaked by LAPSUS$[5].



Ref: virustotal.com

Normally to decode the shellcode our decoder would simply rip out the hexlified data and rebuild it:

```
def decoder(data):
    ret = []
    c2s = []
    blob = re.findall(b'''fce8[0-9a-f]+''', data)
    for val in blob:
```

```
        if len(val)%2 >0:
            val = val[:-1]
        temp = binascii.unhexlify(val)
        s = re.findall(b'[ -~]{4,}', temp)
        t = [x.decode('cp1250') for x in s]
        c2s.append(t[-1])
        ret += t    return((ret,c2s))
```

Since it is UUIDs then the process is still pretty simple but we just need to account for whitespace and then abuse try/catch to end the collection:

```
idx = data.find(b'0089e8fc')
t = data[idx:].split(b'\x00')
tt = [x for x in t if x != b'']
sc = b''
c2s = []
ret = []
for val in tt:
    try:
        u = uuid.UUID(val.decode('cp1250'))
        sc += u.bytes_le
    except:
        break
if sc != '':
    s = re.findall(b'[ -~]{4,}', sc)
    t = [x.decode('cp1250') for x in s]
    c2s.append(t[-1])
    ret += t
```

The debug artifacts in the sample refers to itself as ‘shellcode_uuids’ which can also be used to find another sample in VirusTotal:

```
a59ab6c4e69fafc927f666cf1afb31a2851a392cc74336d8a31e15daf1f343ff
```

The stagers appear to be the same however, so instead of we can search for something different to pivot on:

```
content:"0089e8fc-"
```

This leads to many more samples and since we already have a decoder we can enumerate the IOCs fairly easily for the samples we have accounted for but there are some others mixed in. For example GoLang samples that have out of order UUIDs compared to the other samples:

```
07122c83922c50b386a060d4fbf21433042118fccf5bc678f78acd377d5dccfe
```

To decode we just find the offset reference to the first UUID in the code and then walk the table to rebuild:

```
if b'Go build ID' not in data:
    t = data[idx:].split(b'\x00')
    tt = [x for x in t if x != b'']
else:
    pe = pefile.PE(data=data)
    memdata = pe.get_memory_mapped_image()
    idx = memdata.find(b'0089e8fc')
    imgbase = pe.OPTIONAL_HEADER.ImageBase
    val_to_find = struct.pack('<I', imgbase + idx)
    offset = data.find(val_to_find)
    blob = data[offset:]
    tt = []
    done = False
    while not done:
        temp = struct.unpack_from('<I', blob)[0]
        if temp == 0:
            break
        temp -= imgbase
        val = memdata[temp:temp+36]
        tt.append(val)
        blob = blob[16:]
```

Decoded data:

```
['49.234.114.124'] [';}$u', 'D$$[[aYZQ', ']hnet', 'hwiniThLw&', 'WWWWWh:Vy', 'QQhP', 'SPhW', 'RRRSRP
```

Also some other samples that have their structure of UUIDs reversed:

```
9bfd19e3e459118981bf8ca009acca84f5f079ad3a7baa5eb6a1b20f97d3e922
```

We just need to account for the decoded shellcode being 16 meaning we were at the end and we possibly need to reverse the order:

```
if len(sc) == 16:
    #reversed
    tt = re.findall(b'''[a-f0-9]{8}\-[a-f0-9]{4}\-[a-f0-9]{4}\-[a-f0-9]{4}\-[a-f0-9]{12}''', data)
    sc = b''
    for val in tt[::-1]:
        try:
            u = uuid.UUID(val.decode('cp1250'))
            sc += u.bytes_le
```

```
        except:
            break
```

Decoded data:

```
['47.116.23.73'] [';}$u', 'D$$[[aYZQ', ']hnet', 'hwiniThLw&', 'WWWWWh:Vy', 'SPhW', 'RRRSRPh', 'VhuF'
```

IOCs:

```
144.168.57.182
49.234.114.124
img.googlesoftup.com
7355-120-239-40-185.ngrok.io
132.232.40.201
159.75.127.118
81.68.200.63
150.158.166.73
47.116.23.73
8.142.131.209
172.29.25.27
169.254.41.35
```

User Agents:

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; Avant Browse
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0; ASU2JS)
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; BOIE9;ENIN)
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0; MAARJS)
User-Agent: Microsoft Internet Explorer
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0; MATBJS)
User-Agent: Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Win64; x64; Trident/6.0; Touch)
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0; Touch; MASPJS)
```

Hashes:

```
fc16b66c98a897df46a91a4ac913993ef4ebf440250051fef1d3151d5c9faeab
f175df01577971262f662f15e36c633e0dedd6c0fff9c22bcf5f702ddecc05b7
dd4d27b29f656b5ad1f5cd177b138fe011cc98f5383a102c9a363de0a0021226
dd3416fa926a9801928d8dafd595ce6c3f41ad909ef4c25deb1b9b6a928e3749
```

```
ca9fcd4be0990d19aff9244f47ad05cb033abe4307c51b7b46b635ecca71c3f5
c916d2ee316491ec643b2b029e4e3acced82eeeb6896d42c9e9693e8d0e495ad
c5c586c5c0ad28761dfaef0d049ed4b862be1398ac2fd75cd04e359170777bd8
bc1936c3a33d4e75603341786227bbe8887a19d7b61926ddbe8875fabd1471fe
a59ab6c4e69fafc927f666cf1afb31a2851a392cc74336d8a31e15daf1f343ff
a3b268e6ecc6d2594d2dfdaeb86c25e9baf337f83659d6a718206474b8c416a1
9bfd19e3e459118981bf8ca009acca84f5f079ad3a7baa5eb6a1b20f97d3e922
8cd86fa95c702dca17ea75893f27ac414e1947fc90ca86bcd38d209ede2ee19b
8c80fa876b01544e85b8588209cefc99827f1a882580778b87274d157b88eda6
8869be33e23370d19c17f33cb5b2e1501b95eafcfd9b71b34bc338f477bc5a3e
80907d4e3519cf212e755b4cc175ce807cf2313ad16d85020f985670be87a4c9
804c3e3709e35949311bae50a1d98d8233c26cf193ff27d439d9da025ab6f94f
6cbe5b78e2c82b3e98fdff531aaf49dd8a7191015af8eb63366eb468693a0532
6237bbef07de46d11d45b4997fc661114f092abcbbe1108b0f6d7508e55e4398
61f47f9b7f68ac996bc3a56c16f7dd4aedbeba6a5d7b46e8406505fc6903524c
588454f02cba40703ed7f6dfe256ecaba3c41bbd80ed1cc0e34ba8d2214c05d0
57c1c1e2f214f4a933fa356219acaec3bb338e6fb15dd6349e6cf04a5289a2e4
53c337b5f52ac1d21ba226e4d027f01a80cc6176cb1d3f1ad1148cf37c8c75eb
488d4cbe017c493c66a769ccb203b40cbca3396c654bfff72c1aebeb41f23297
47df2d7eb512a533511104c939bfa89e6266d3ce9f0ffdced681768e183244a6
44d6dd82e5b7ca8976c50c9165ce822fbef86075a7bde43cfd364e28f30e400c
42d9638489f248f1177b86739d1d9ce2a9f4aa591523cd7655b324853895e857
3ba3aaecbc834da3dba0ec5d20d5c9cb119f27bc660f720f6cc3da929da0e529
398b4205fe8f4d1bf005782bd05730c45c39224c14b2878865dd3d7a255cd20b
38f4167638607dd6a299d3c02ceb984fc4f9d7491d4c25a609186e582e705505
317dc94e847fec42f101709043eff4d705b531b1e1d433f7af1f4ef62185e194
2cfdacdc363ddf146d59add3bf49b49a29df951ff3be4bf35e70c2125b2751d1
2c3db921dd86b801b9e856c28830990311db52b18f3a1f4a100eb84d3aaf55a7
24d66bc81ac389c5580a7de37f531cd89e5dbf97f3048e5164a1cb829d16c51b
24b60dfda87cebafec57f827b0fc534bc3fe3afa178b5f15683fe30bcf4041fe
23084aa205a2a09623752e0fa807c722a9a23c169eb5f7a04fb50a8c4f7a6924
1322219fff925f6081bb38c09d8215b796fbc910003a2df2ad627558479f2cf8
07122c83922c50b386a060d4fbf21433042118fccf5bc678f78acd377d5dccfe
033b1cad6953ad623cbc565fa7afb4751cac1b25f5050dab08646faafff29619
```

YARA:

```
rule cs_hexlified_stager_sc
{
strings:
$a1 = "d2648b52308b" nocase
condition:
all of them
}
```

References

1:https://medium.com/walmartglobaltech/investigation-into-the-state-of-nim-malware-14cc543af811

Get Jason Reaves's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

2:https://medium.com/walmartglobaltech/trickbot-crews-new-cobaltstrike-loader-32c72b78e81c

3:https://medium.com/walmartglobaltech/investigation-into-the-state-of-nim-malware-part-2-a28bffffa671

4:https://en.wikipedia.org/wiki/Universally_unique_identifier

5:https://threatpost.com/nvidias-stolen-code-signing-certs-sign-malware/178784/
