POS Malware Used at Fuel Pumps

norfolkinfosec.com/pos-malware-used-at-fuel-pumps/

norfolk December 23, 2019

In December 2019, VISA Security released a bulletin detailing multiple incidents in which threat actors targeted point of sale systems used at fuel dispensing companies with malware designed to parse out credit card numbers from these systems. This blog post examines a file, 19d38325f715f623bd4b6e819a150cde, associated with the first of three listed incidents in that bulletin.

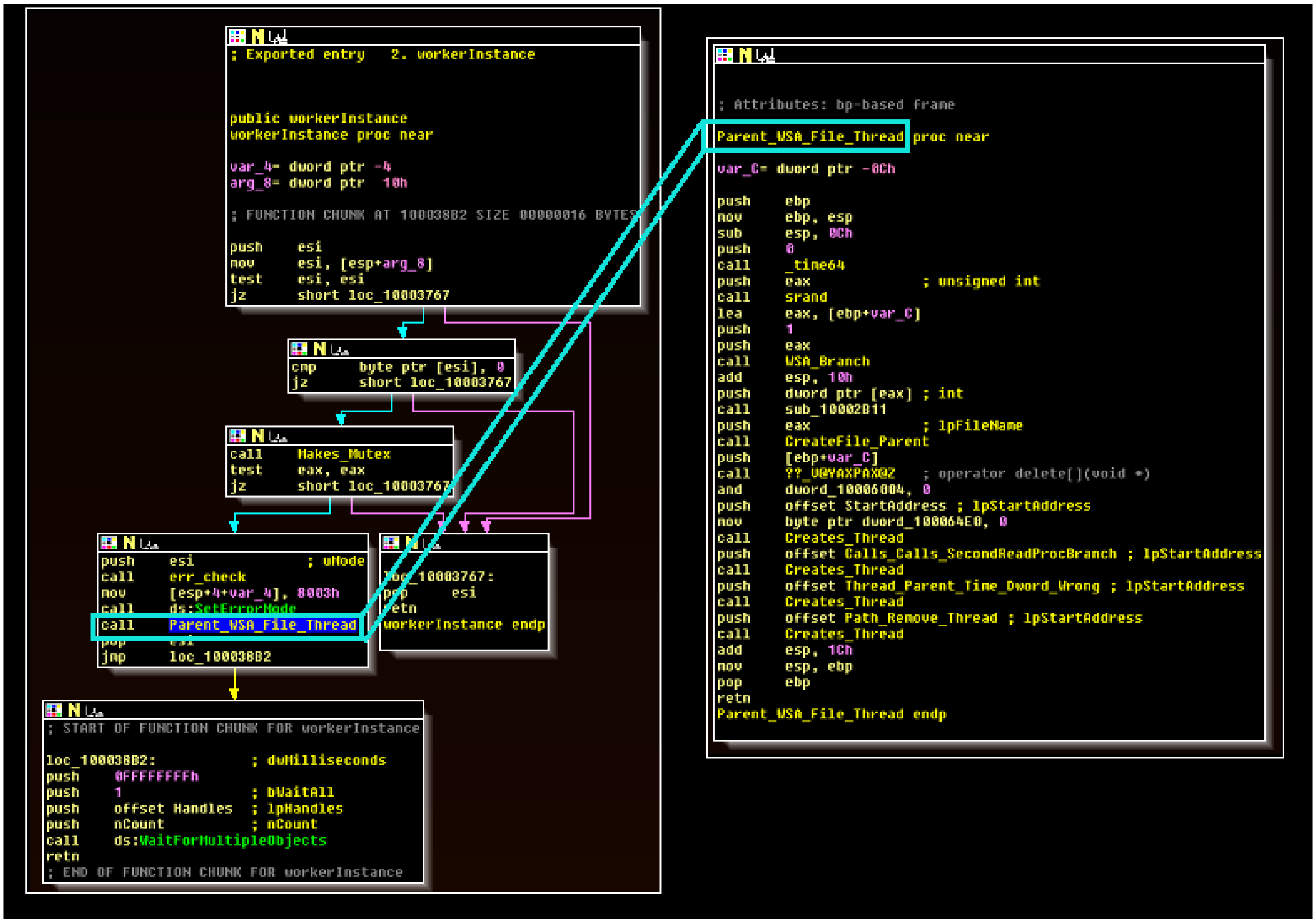
There are several notable characteristics regarding this malware, including a unique way for the threat actors to terminate the tool.

MD5: 19d38325f715f623bd4b6e819a150cde
SHA1: 81c4a8cf8c0da1c590377b37ed5cff8771560a3d
SHA256: 7a207137e7b234e680116aa071f049c8472e4fb5990a38dab264d0a4cde126df

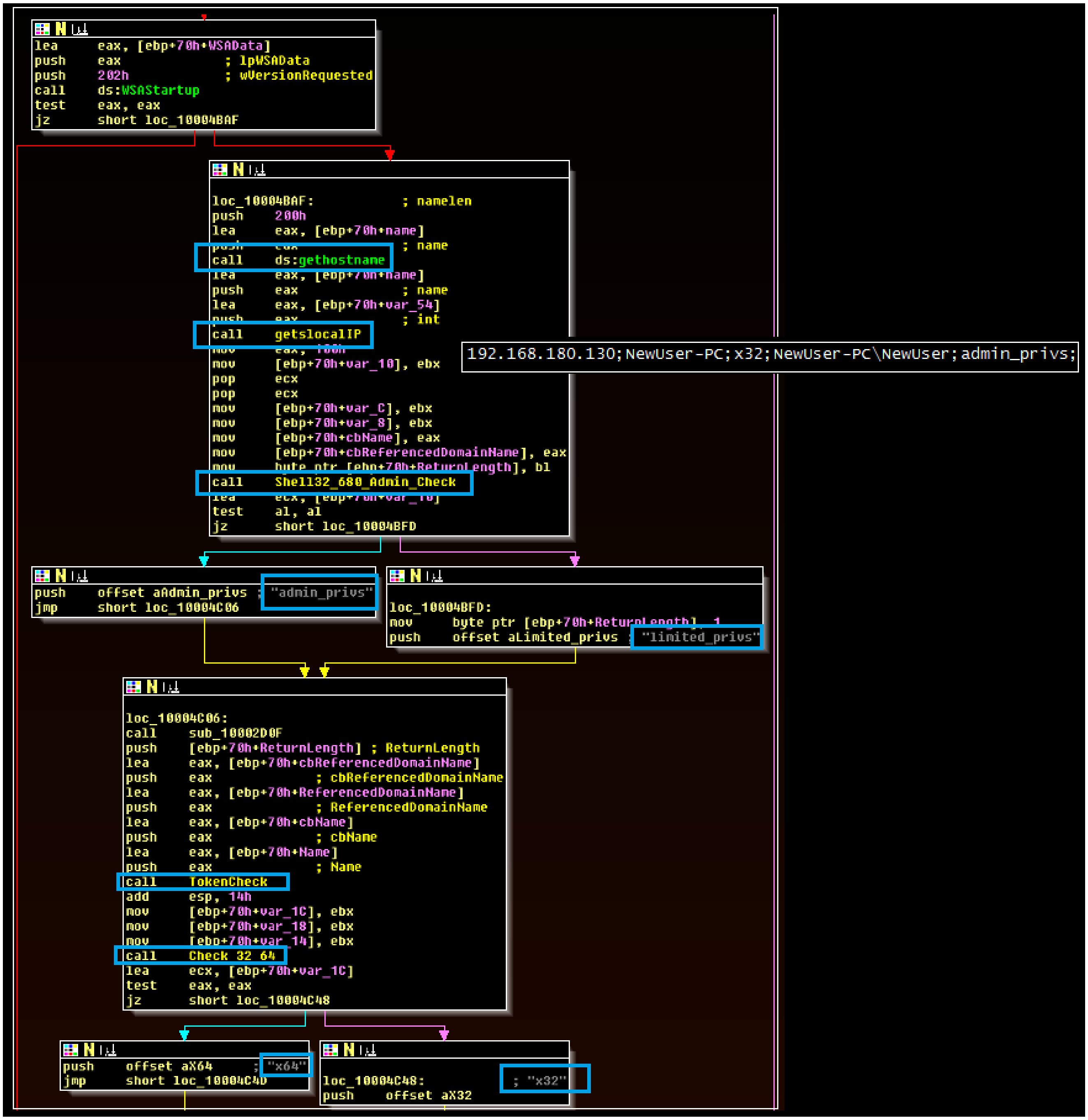
The file appears to be a variant of the Grateful/Framework POS family. While this variant (via a similar file, 0EB7AC6D2D99D702ECC8B86FF90B0AAC) are described elsewhere, this blog is currently unable to replicate or identify the data exfiltration method detailed in external posts. This method appears statically in strings in similar – but larger – samples, suggesting that it may have actually been removed for certain variants. If that is the case, it would also imply that the threat actors exifiltrated the data through other malware or tools, which would be consistent with some vendor observations. Further discussion around this point and the discrepancies in reported functionality around these hashes can be found in a later section.

The file contains two exports:
– workerInstance (main functionality)
– debugPoint (enters a sleep loop)

The workerInstance export is used to launch the main functionality of the malware. In addition, the malware also expects to receive a file path as an argument. When this export is called, the malware creates a mutex named "Global.Ms.ThreadPooling.MyAppSingleInstance" and then collects local data about the infected workstation. This data is written to the filepath specified at runtime.



; Exported entry 2. workerInstance
public workerInstance
workerInstance proc near
var_4= dword ptr -4
arg_8= dword ptr 10h
; FUNCTION CHUNK AT 100038B2 SIZE 00000016 BYTES
push esi
mov esi, [esp+arg_8]
test esi, esi
jz short loc_10003767
cmp byte ptr [esi], 0
jz short loc_10003767
call Makes_Mutex
test eax, eax
jz short loc_10003767
push esi ; uMode
call err_check
mov [esp+4+var_4], 8003h
call ds:SetErrorMode
call Parent_WSA_File_Thread
pop esi
jmp loc_100038B2
loc_10003767:
pop esi
retn
workerInstance endp
; START OF FUNCTION CHUNK FOR workerInstance
loc_100038B2: ; dwMilliseconds
push 0FFFFFFFFh
push 1 ; bWaitAll
push offset Handles ; lpHandles
push nCount ; nCount
call ds:WaitForMultipleObjects
retn
; END OF FUNCTION CHUNK FOR workerInstance
; Attributes: bp-based frame
Parent_WSA_File_Thread proc near
var_C= dword ptr -0Ch
push ebp
mov ebp, esp
sub esp, 0Ch
push 0
call _time64
push eax ; unsigned int
call srand
lea eax, [ebp+var_C]
push 1
push eax
call WSA_Branch
add esp, 10h
push dword ptr [eax] ; int
call sub_10002B11
push eax ; lpFileName
call CreateFile_Parent
push [ebp+var_C]
call ??_V@YAXPAX@Z ; operator delete[](void *)
and dword_10006084, 0
push offset StartAddress ; lpStartAddress
mov byte ptr dword_100064E0, 0
call Creates_Thread
push offset Calls_Calls_SecondReadProcBranch ; lpStartAddress
call Creates_Thread
push offset Thread_Parent_Time_Dword_Wrong ; lpStartAddress
call Creates_Thread
push offset Path_Remove_Thread ; lpStartAddress
call Creates_Thread
add esp, 1Ch
mov esp, ebp
pop ebp
retn
Parent_WSA_File_Thread endp



The malware runs four threads:

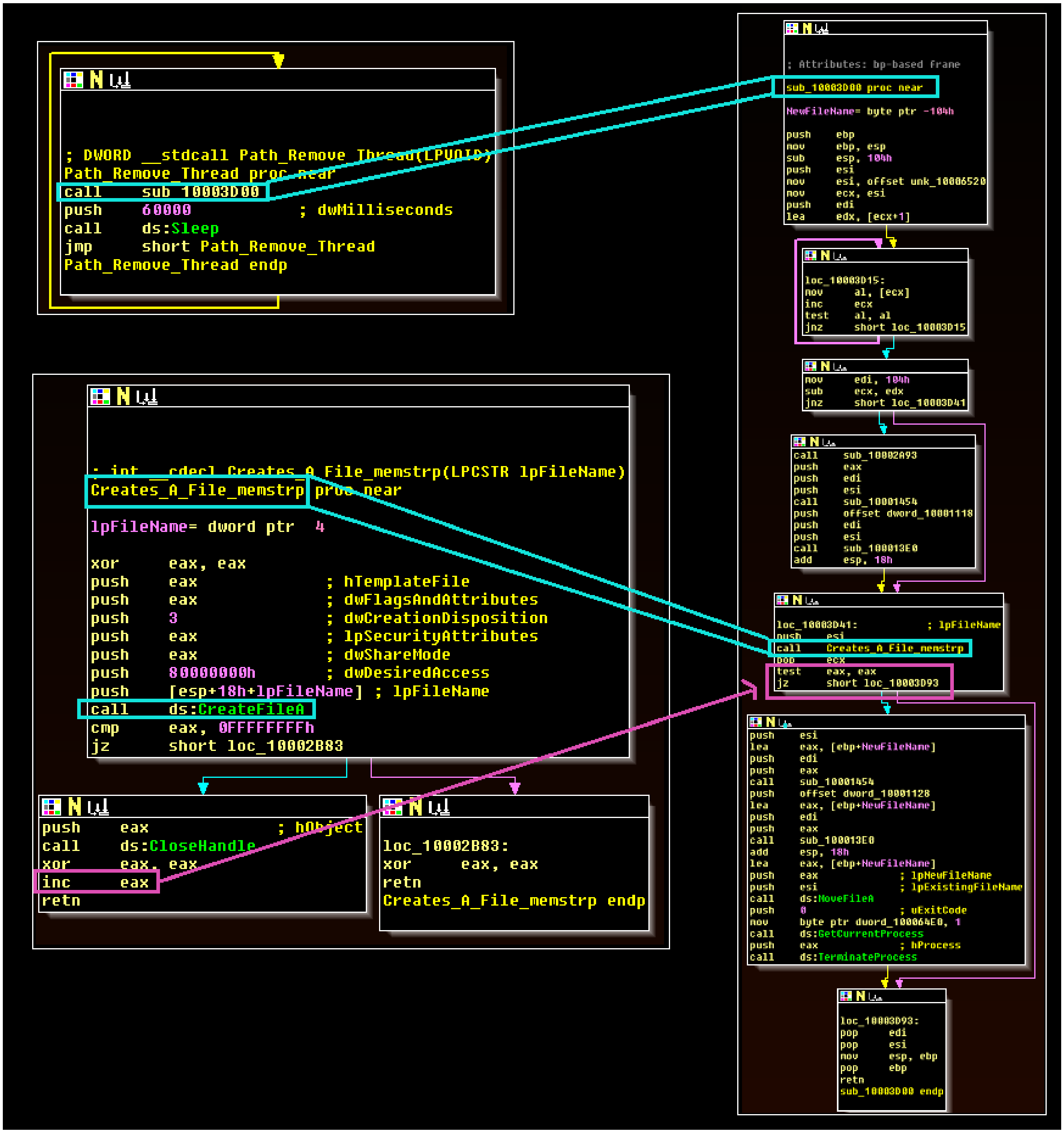
– Thread 1: Enters memory scraping loop
– Thread 2: Enters memory scraping loop
– Thread 3: Checks length of process to be scraped. Process must be > 4 characters.
– Thread 4: Terminates the malware if a “stopper” file is found in the working directory

Of these, **Thread 4** is among the most novel and allows the threat actors to terminate the malware. The malware takes the filename “memscrp.stp” and appends it to a string containing the working directory of the DLL. The malware will then use the CreateFile API to try to access a file with the name at this location. It then performs a comparison:

1) If the CreateFile call generated an error (i.e. the file was not present at the time of the check), EAX is zeroed out and the routine sleeps for sixty seconds before trying again.

2) If this call does *not* generate an error (i.e. the file exists), the malware uses the MoveFile API call to add a .stopped extension to this file and then terminates.

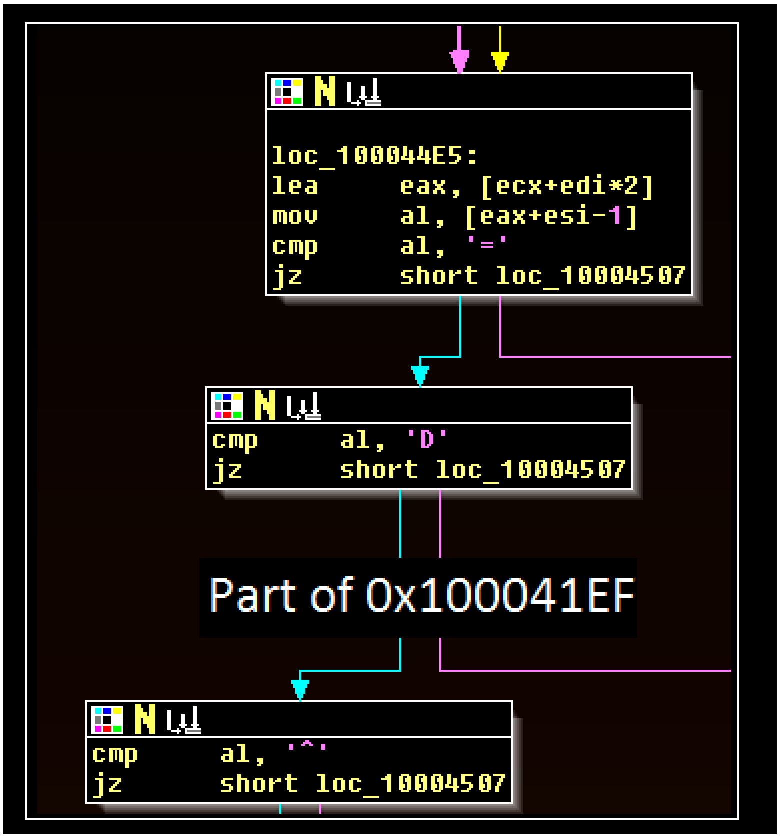
This workflow is shown below.



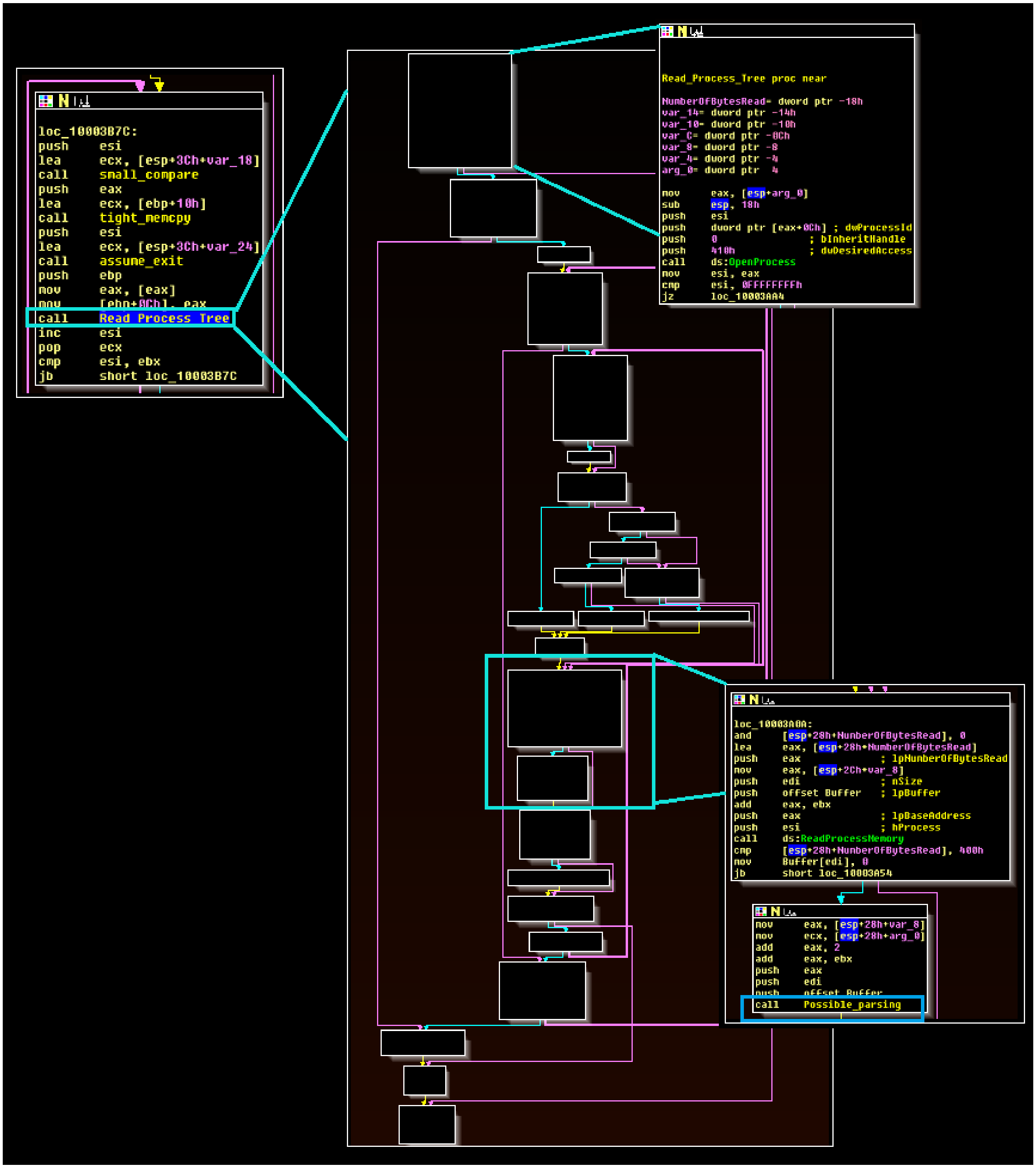
The advantages to this are unclear; however, one possibility is that this approach allows the threat actors to terminate the malware without the need for command and control implementation.

Memory Scraping Threads

As noted in another blog post, this malware forgoes more targeted scraping (in which specific BINs are selected) in favor of a broader collection. The threat actors' scraping logic is not yet fully understood; however, several characteristics of credit card track data do appear, including the common "=" and "^" separators:

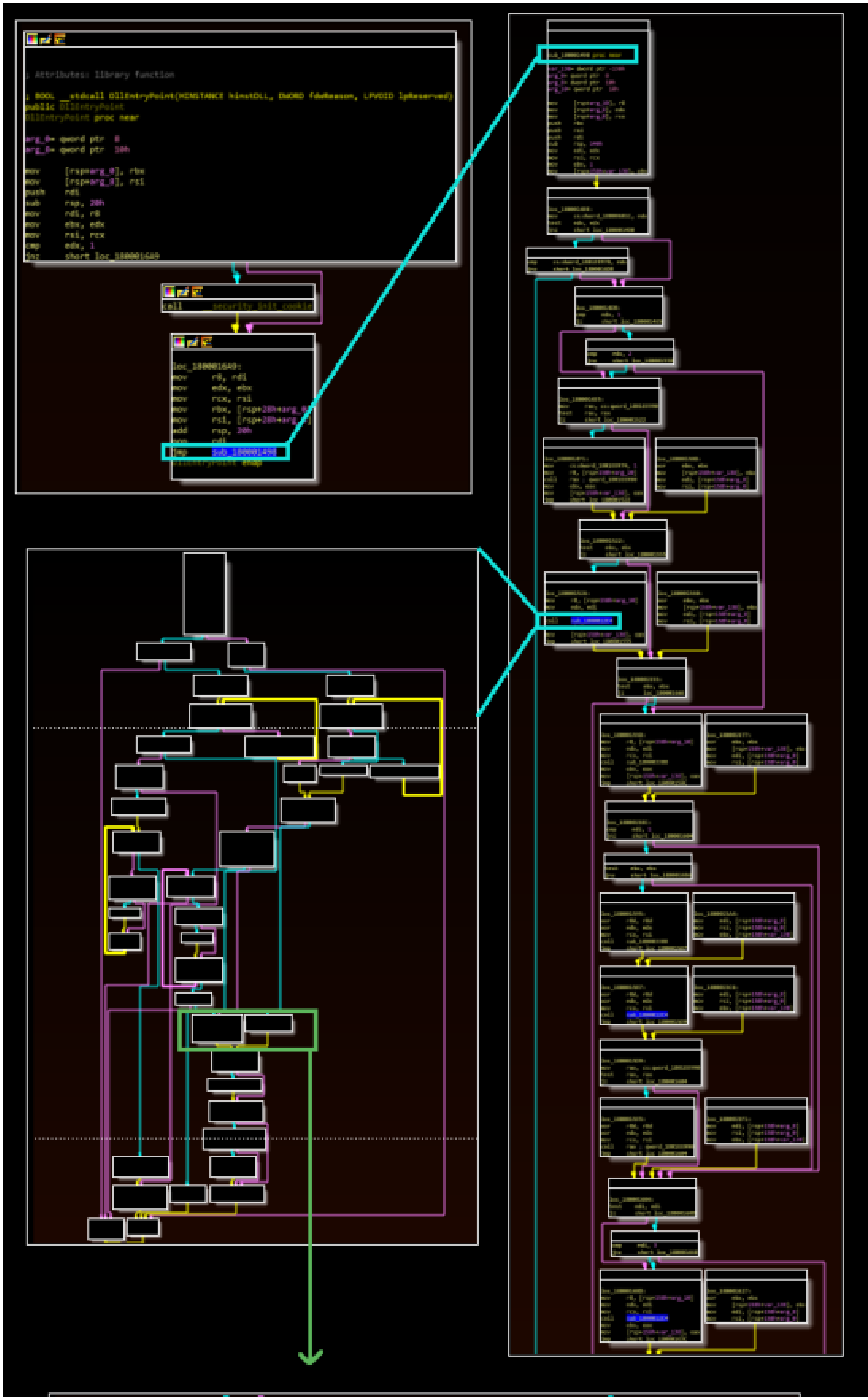


The scraping threads use the ReadProcessMemory API call to run data from all of the processes on the infected system. Unlike previously documented samples, no apparent whitelisting was present in the malware during static analysis, and during dynamic runs of the malware the scraper searched for data without discretion. The comparison logic in the image above takes place within the "Possible_Parsing" function boxed in blue at the bottom right of the image below:



At this stage, this blog **has not** identified where this data is stored or how it is transmitted. While some variants of this file have C2 functionality via DNS requests (a previously known and documented feature), such features appear absent from the file analyzed here and reported by VISA. This blog also performed a dynamic comparison between a known DNS variant and the file analyzed here using test data. The DNS variant immediately began communicating with external servers (including a public IP checker and the C2 server) and eventually attempted to transmit scraped test data over the DNS protocol. The file analyzed in this blog post did not perform these tasks.

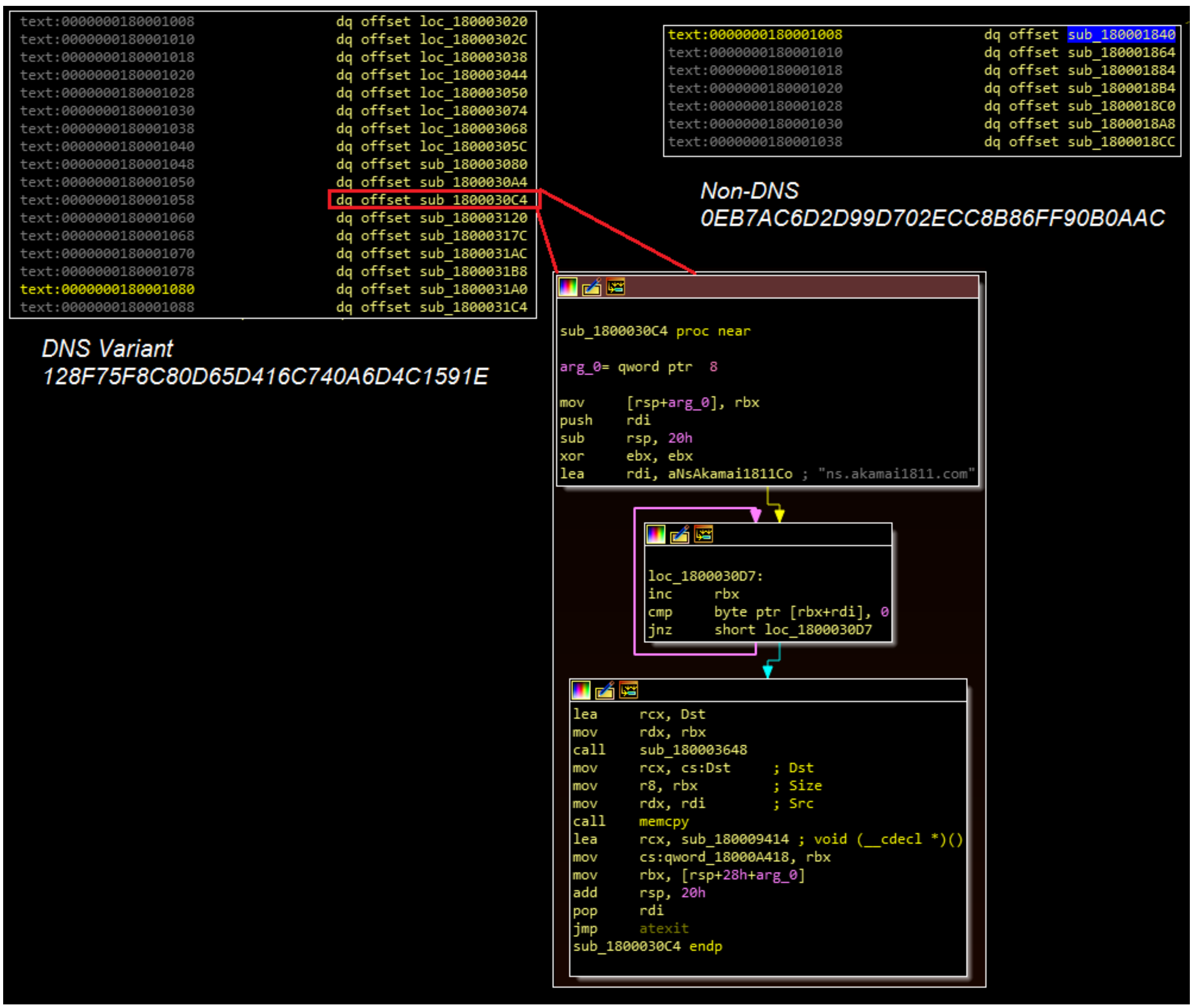
A static comparison of both variants, with a focus on the DNS variant’s C2 server, shows that both files have nearly identical code leading to where this server is referenced in the DNS version and where one would expect it to be referenced in the non-DNS version:



This code

workflow is nearly identical in both variants

However, examining this location (boxed in orange above) shows that several functions are not present in the non-DNS version. Most importantly, none of the functions in this location contain code matching the routine with the C2 reference in the DNS version:



The DNS variant (top left) contains additional functionality not present in the non-DNS variant

If these features have been removed, this blog postulates that either a file saving mechanism exists but has not yet been identified, or an additional file is used to run the DLL and collect data.

Additional Variants

As noted above, there are other variants of this scraper. A VirusTotal pivot on the workerInstance export identifies eight total samples, with varying compile times. Of these samples, some feature DNS exfiltration capabilities and others do not:

Non-DNS
32ccf851b0b81252aa2bfdf2e8b416cb Compilation Timestamp: 2018-12-10 20:06:42 (27KB)
0eb7ac6d2d99d702ecc8b86ff90b0aac Compilation Timestamp: 2019-04-11 13:26:51 (27kB)
576039d7cb54b749af5ed3d3558ee296 Compilation Timestamp: 2018-11-07 11:56:06 (25KB)
19d38325f715f623bd4b6e819a150cde Compilation Timestamp: 2018-12-10 20:07:02 (23KB) (blog version)

DNS
0576380f93f49279491177d96d84ad7e Compilation Timestamp: 2018-11-27 20:06:19 (89Kb)
353b0df3a9efce2d32f6097cab8fffc3 Compilation Timestamp: 2018-11-27 20:06:44 (46KB)
128f75f8c80d65d416c740a6d4c1591e Compilation Timestamp: 2018-11-27 20:06:19(44KB)
4ed6cc403d5ea6abae458ba6f43ad4f3 Compilation Timestamp: 2018-11-27 20:06:44 (42KB)

Interestingly, the DNS variants were all compiled within a minute of each other. While two files share the same timestamp (and perhaps are the same file, dumped from memory or disk differently), there are still three unique timestamps from this set. In addition, these files are noticeably larger than the apparent non-DNS version. With one exception, these files also have compilation timestamps predating the non-DNS versions, although this data set might not be complete given the limitations in VirusTotal's search range (although none of the DNS versions with this data query had compilation timestamps beyond 2018).

One possible explanation is that the threat actor shifted away from DNS exfiltration in favor of a quiet collection or the use of an external tool. Another possibility is that the tool is shared across multiple threat actors with different operational behaviors. The short window of compilation timestamps for the DNS samples could represent different builds for multiple simultaneous targets, threat actor testing, or a more benign explanation.

The DNS versions all use "ns.akamai1811.com" as their C2.