Weaponized Disk Image Files: Analysis, Trends and Remediation

crowdstrike.com/blog/weaponizing-disk-image-files-analysis/

Guillermo Taibo

May 26, 2020

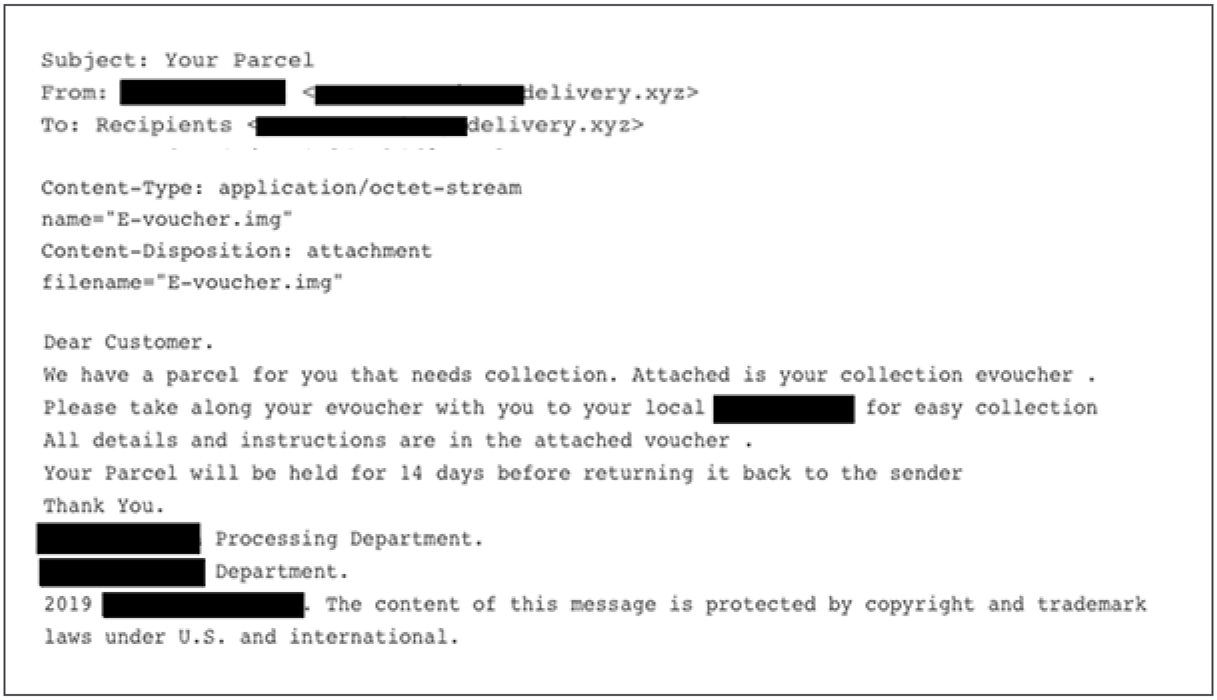
Throughout 2019 and the beginning of 2020, the CrowdStrike® Falcon Complete^{TM} team continuously observed a spike in the delivery of weaponized disk image files. Files such as ISO and IMG were sent to infect systems with the goal of delivering remote access trojans (RATs) as well as a few other malware variants. We've identified that these files are typically delivered via phishing campaigns as an attachment or link — a malicious URL in the body of the email or within crack software downloads.

Cyber criminals have been taking advantage of built-in Windows capabilities to mount disk image files once they are opened by the end user. There are multiple disk image file formats, but we have seen ISO and IMG files being abused the most. A disk image is essentially a virtual copy of a physical disk that houses all of the files and requires that it be mounted in order to access its contents. The advantages of using disk images, combined with the easy access to purchasing RATs, make this a preferred and effective method for cybercriminals.

In this blog, I dissect a campaign that uses this method to compromise a system, providing insight into what the CrowdStrike FalconComplete team has observed since 2019. I will also provide step-by-step remediation along with recommendations for how to implement this approach in your network.

Parcel-themed Phishing Email Scenario

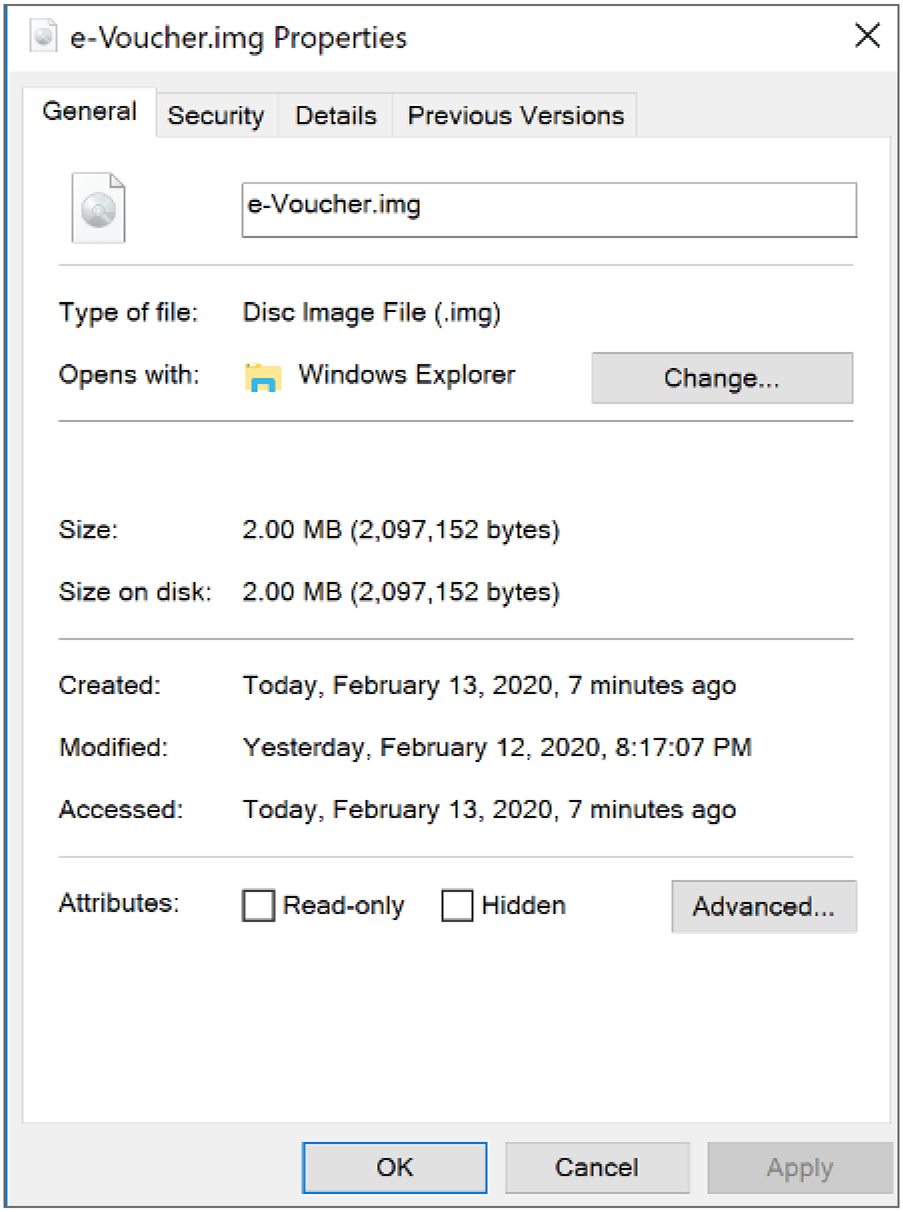
The chain starts with a simple email containing a disk image file (.IMG) to socially engineer the victim into viewing the contents. The message seems to be coming from a worldwide package delivery company.



```
Subject: Your Parcel
From: [redacted] <[redacted]delivery.xyz>
To: Recipients <[redacted]delivery.xyz>

Content-Type: application/octet-stream
name="E-voucher.img"
Content-Disposition: attachment
filename="E-voucher.img"

Dear Customer.
We have a parcel for you that needs collection. Attached is your collection evoucher .
Please take along your evoucher with you to your local [redacted] for easy collection
All details and instructions are in the attached voucher .
Your Parcel will be held for 14 days before returning it back to the sender
Thank You.
[redacted] Processing Department.
[redacted] Department.
2019 [redacted]. The content of this message is protected by copyright and trademark
laws under U.S. and international.
```

Figure 1. Phishing contents sample. The delivery company did not send this email.

The attachment in this sample is only 2MB, which raises a flag immediately as disk images are typically larger in size.



e-Voucher.img Properties

e-Voucher.img

Type of file: Disc Image File (.img)

Opens with: Windows Explorer

Size: 2.00 MB (2,097,152 bytes)

Size on disk: 2.00 MB (2,097,152 bytes)

Created: Today, February 13, 2020, 7 minutes ago

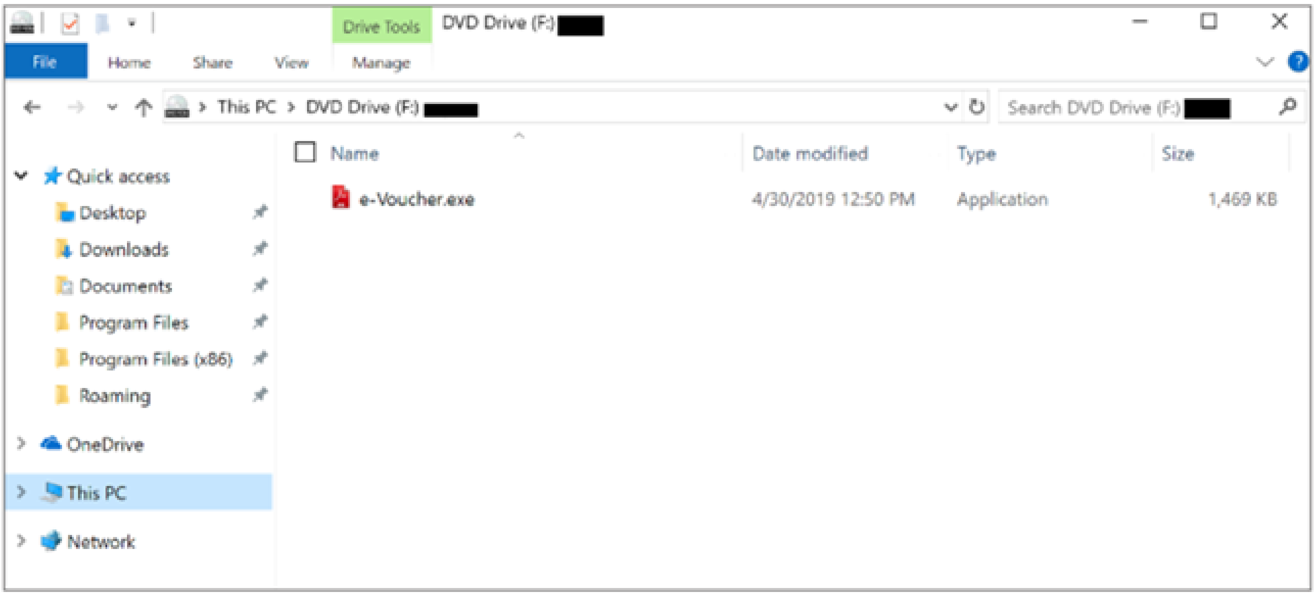
Modified: Yesterday, February 12, 2020, 8:17:07 PM

Accessed: Today, February 13, 2020, 7 minutes ago

Attributes: □ Read-only □ Hidden

Figure 2. IMG file properties

Double-clicking on the file allows Windows 8 and Windows 10 to mount the IMG file natively to the next available drive. This sample uses a PDF icon as a disguise.

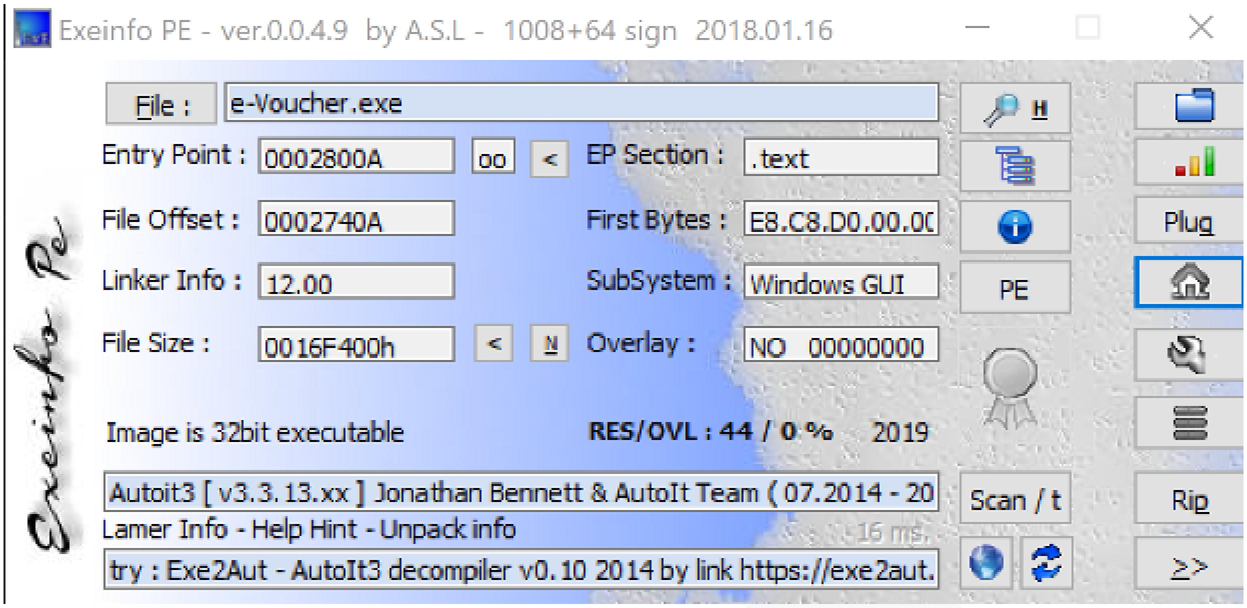


Name	Date modified	Type	Size
e-Voucher.exe	4/30/2019 12:50 PM	Application	1,469 KB

Figure 3. IMG file mounted on disk

Analysis

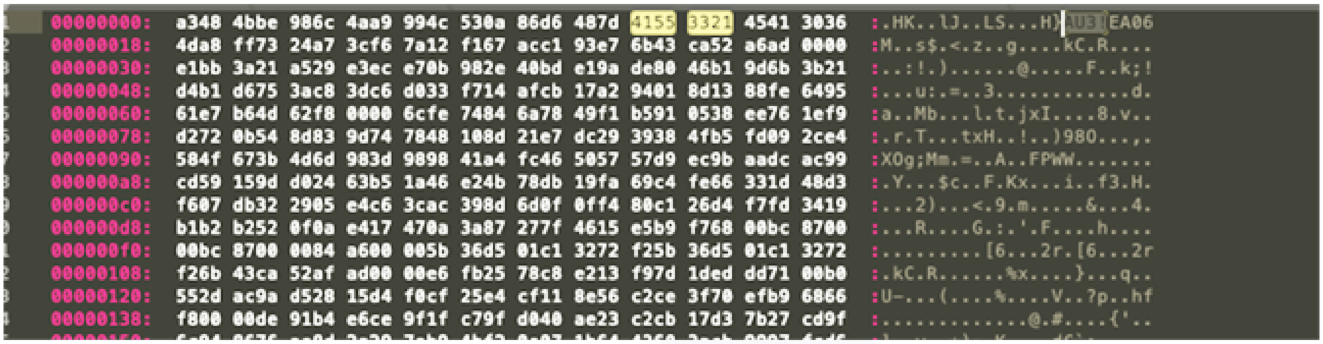
Exeinfo PE identified the binary as a compiled AutoIT script version 3. AutoIT is a scripting language used to automate Windows GUI tasks. Cybercriminals would first compile these scripts into an executable using the Aut2Exe compiler and further convert it into a disk image file to then distribute it widely in campaigns.



File : e-Voucher.exe
Entry Point : 0002800A
EP Section : .text
File Offset : 0002740A
First Bytes : E8.C8.D0.00.0C
Linker Info : 12.00
SubSystem : Windows GUI
File Size : 0016F400h
Overlay : NO 00000000
Image is 32bit executable
RES/OVL : 44 / 0 % 2019
Autoit3 [v3.3.13.xx] Jonathan Bennett & AutoIt Team (07.2014 - 20
Lamer Info - Help Hint - Unpack info
16 ms.
try : Exe2Aut - AutoIt3 decompiler v0.10 2014 by link https://exe2aut.

Figure 4. Exeinfo PE against binary `e-voucher.exe`

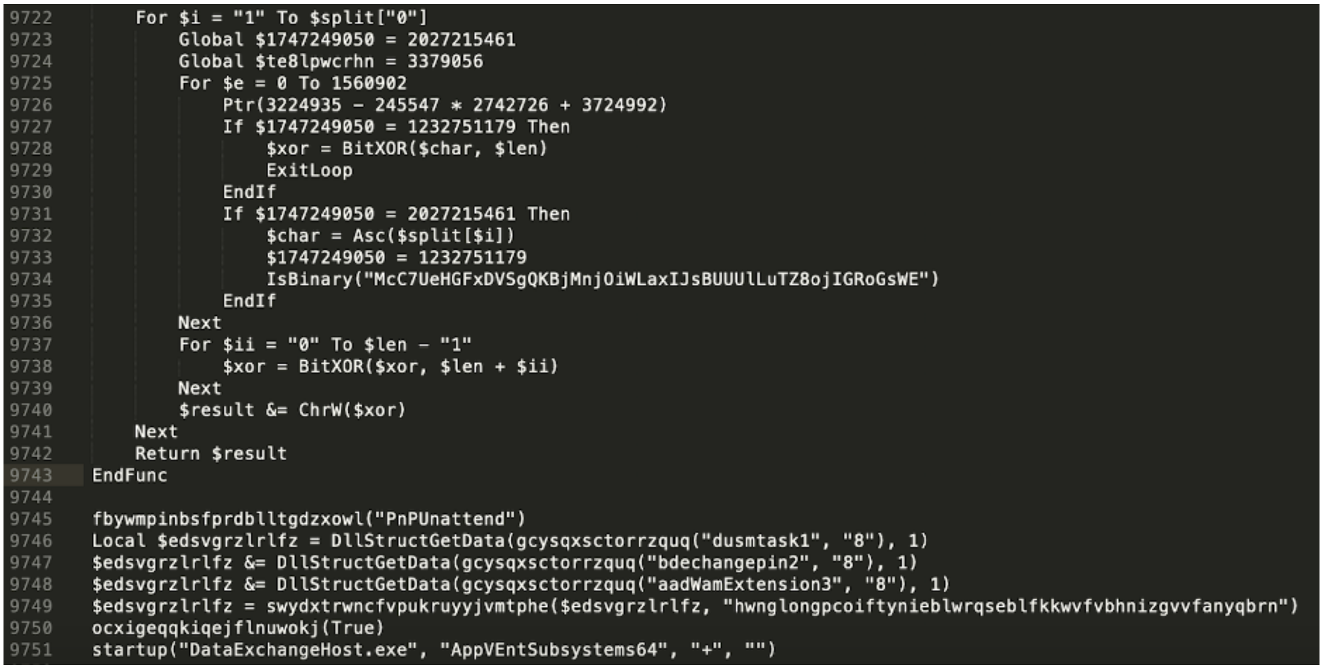
Dumping the rcdata resource and reviewing the strings shows AU3!, a common string seen in AutoIT-developed scripts.



```
00000000:  a348 4bbe 986c 4aa9 994c 530a 86d6 487d 4155 3321 4541 3036  :.HK..lJ..LS...H}AU3!EA06
00000018:  4da8 ff73 24a7 3cf6 7a12 f167 acc1 93e7 6b43 ca52 a6ad 0000  :M..s$.<.z..g.....kC.R....
00000030:  e1bb 3a21 a529 e3ec e70b 982e 40bd e19a de80 46b1 9d6b 3b21  :..:!.).......@.....F..k;!
00000048:  d4b1 d675 3ac8 3dc6 d033 f714 afcb 17a2 9401 8d13 88fe 6495  :...u:.=..3.............d.
00000060:  61e7 b64d 62f8 0000 6cfe 7484 6a78 49f1 b591 0538 ee76 1ef9  :a..Mb...l.t.jxI....8.v..
00000078:  d272 0b54 8d83 9d74 7848 108d 21e7 dc29 3938 4fb5 fd09 2ce4  :.r.T...txH..!..)98O...,.
00000090:  584f 673b 4d6d 983d 9898 41a4 fc46 5057 57d9 ec9b aadc ac99  :XOg;Mm.=..A..FPWW.......
000000a8:  cd59 159d d024 63b5 1a46 e24b 78db 19fa 69c4 fe66 331d 48d3  :.Y...$c..F.Kx...i..f3.H.
000000c0:  f607 db32 2905 e4c6 3cac 398d 6d0f 0ff4 80c1 26d4 f7fd 3419  :...2)...<.9.m.....&...4.
000000d8:  b1b2 b252 0f0a e417 470a 3a87 277f 4615 e5b9 f768 00bc 8700  :...R....G.:.'.F....h....
000000f0:  00bc 8700 0084 a600 005b 36d5 01c1 3272 f25b 36d5 01c1 3272  :..........[6...2r.[6...2r
00000108:  f26b 43ca 52af ad00 00e6 fb25 78c8 e213 f97d 1ded dd71 00b0  :.kC.R......%x....}...q..
00000120:  552d ac9a d528 15d4 f0cf 25e4 cf11 8e56 c2ce 3f70 efb9 6866  :U-...(....%....V..?p..hf
00000138:  f800 00de 91b4 e6ce 9f1f c79f d040 ae23 c2cb 17d3 7b27 cd9f  :.............@.#....{'..
```

Figure 5. Hexdump of `e-voucher.exe`

The AutoIT script is obfuscated, and it is used as a dropper to eventually load the NanoCore RAT on the intended system.



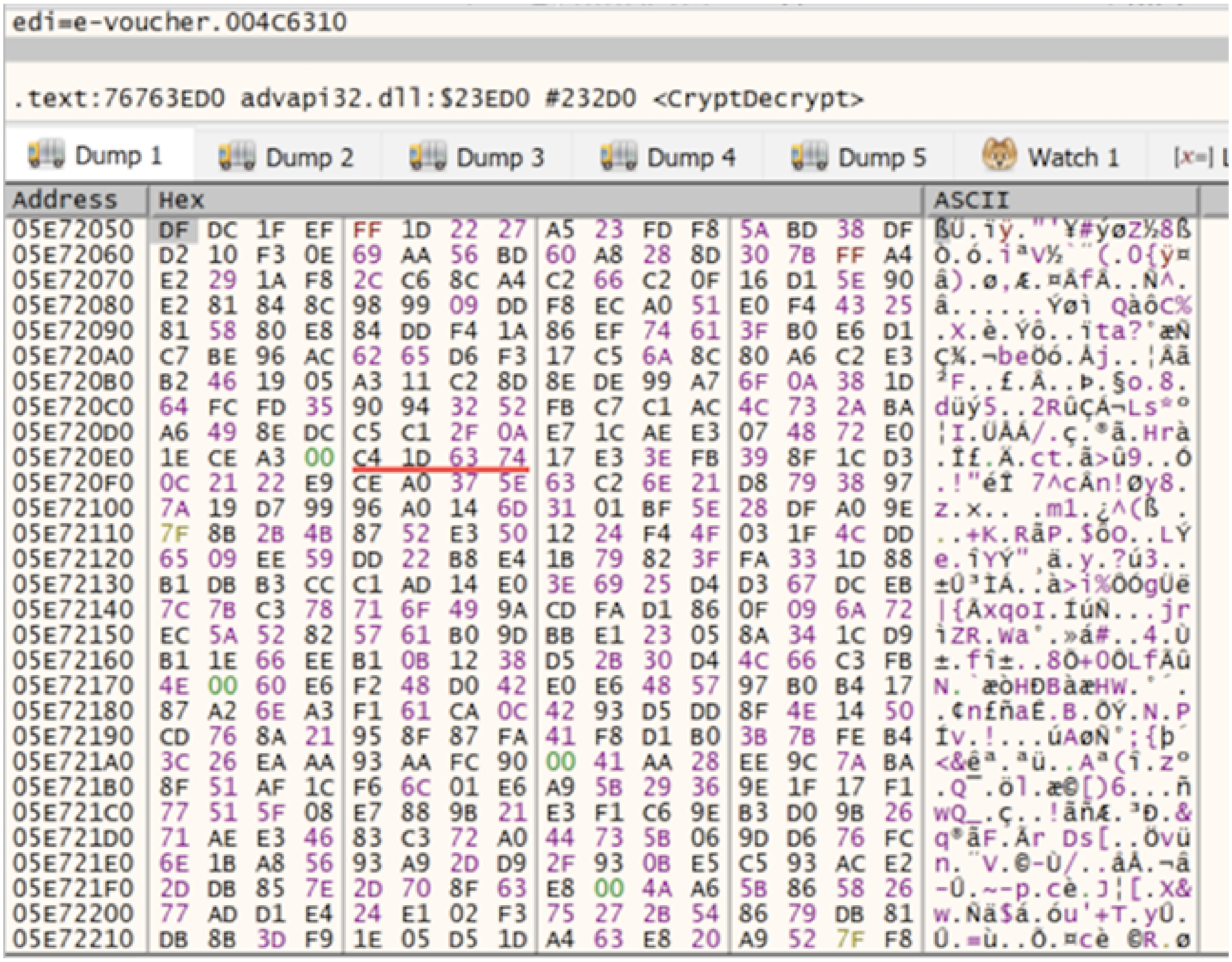
```
9722        For $i = "1" To $split["0"]
9723            Global $1747249050 = 2027215461
9724            Global $te8lpwcrhn = 3379056
9725            For $e = 0 To 1560902
9726                Ptr(3224935 - 245547 * 2742726 + 3724992)
9727                If $1747249050 = 1232751179 Then
9728                    $xor = BitXOR($char, $len)
9729                    ExitLoop
9730                EndIf
9731                If $1747249050 = 2027215461 Then
9732                    $char = Asc($split[$i])
9733                    $1747249050 = 1232751179
9734                    IsBinary("McC7UeHGFxDVSgQKBjMnjOiWLaxIJsBUUUlLuTZ8ojIGRoGsWE")
9735                EndIf
9736            Next
9737            For $ii = "0" To $len - "1"
9738                $xor = BitXOR($xor, $len + $ii)
9739            Next
9740            $result &= ChrW($xor)
9741        Next
9742        Return $result
9743    EndFunc
9744
9745    fbywmpinbsfprdblltgdzxowl("PnPUnattend")
9746    Local $edsvgrzlrlfz = DllStructGetData(gcysqxsctorrzquq("dusmtask1", "8"), 1)
9747    $edsvgrzlrlfz &= DllStructGetData(gcysqxsctorrzquq("bdechangepin2", "8"), 1)
9748    $edsvgrzlrlfz &= DllStructGetData(gcysqxsctorrzquq("aadWamExtension3", "8"), 1)
9749    $edsvgrzlrlfz = swydxtrwncfvpukruyyjvmtphe($edsvgrzlrlfz, "hwnglongpcoiftynieblwrqseblfkkwvfvbhnizgvvfanyqbrn")
9750    ocxigeqqkiqejflnuwokj(True)
9751    startup("DataExchangeHost.exe", "AppVEntSubsystems64", "+", "")
```

Figure 6. Snippet of obfuscated AutoIt script

Beginning on line 9746 in Figure 6, we can see the following three resources:

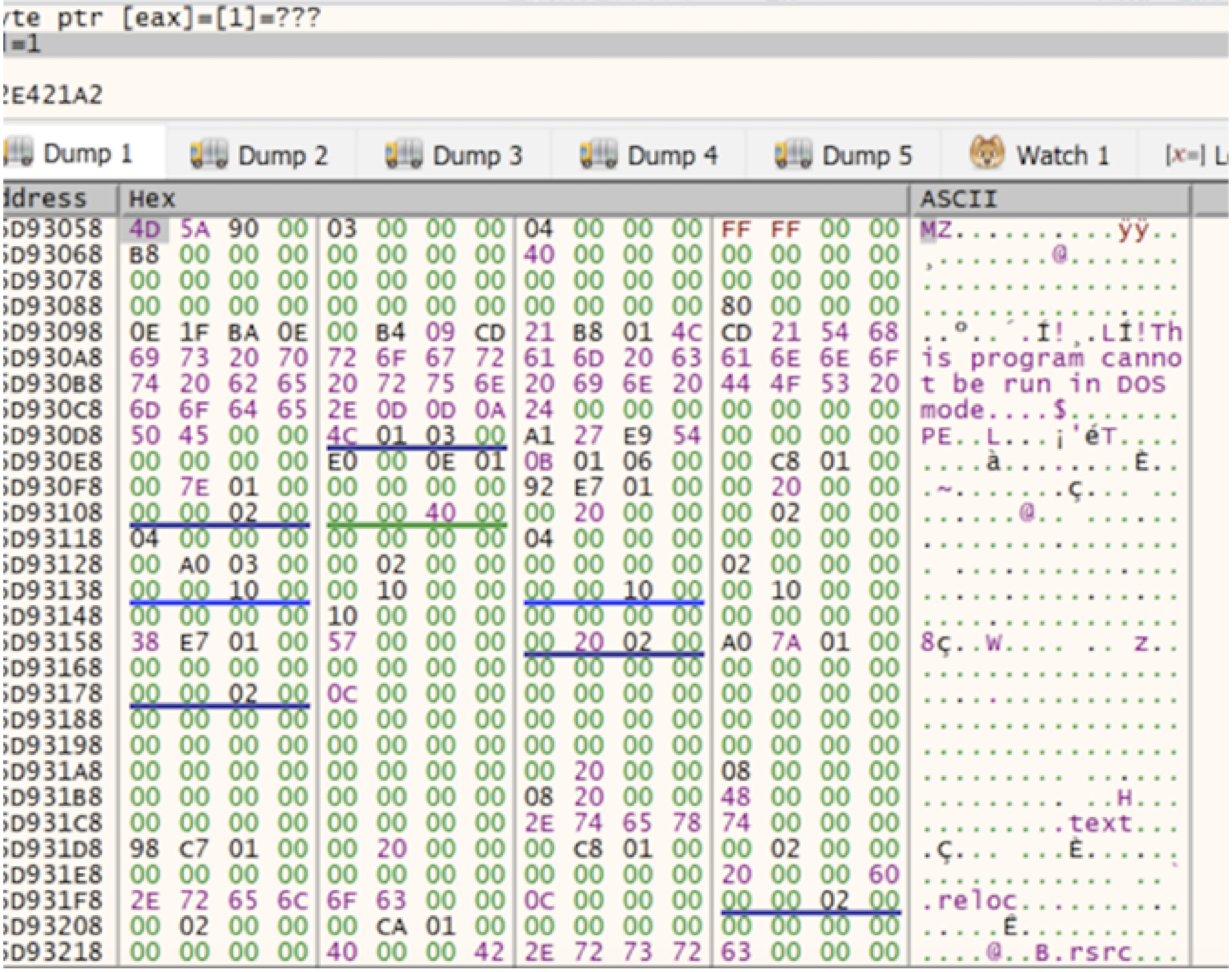
```
dusmtask1
bdechangepin2
aadWamExtension3
```

The script merges these three resources and passes the key “hwnglongpcoiftynieblwrqseblfkkwvfvbhnizgvvfanyqbrn” as the second parameter to the function swydxtrwncfvpukruyyjvmtphe(). To decrypt, it creates a hash using CryptCreateHash with this key. Consequently, it then uses the function CryptDeriveKey and creates a separate key from the results of CryptCreateHash. Finally, CryptDecrypt is used to decrypt the resource.



```
edi=e-voucher.004C6310

.text:76763ED0 advapi32.dll:$23ED0 #232D0 <CryptDecrypt>

Address   Hex                                              ASCII
05E72050  DF DC 1F EF FF 1D 22 27 A5 23 FD F8 5A BD 38 DF  ßÜ.ïÿ."'¥#ýøZ½8ß
05E72060  D2 10 F3 0E 69 AA 56 BD 60 A8 28 8D 30 7B FF A4  Ò.ó.iªV½`¨(.0{ÿ¤
05E72070  E2 29 1A F8 2C C6 8C A4 C2 66 C2 0F 16 D1 5E 90  â).ø,Æ.¤ÂfÂ..Ñ^.
05E72080  E2 81 84 8C 98 99 09 DD F8 EC A0 51 E0 F4 43 25  â......Ýøì QàôC%
05E72090  81 58 80 E8 84 DD F4 1A 86 EF 74 61 3F B0 E6 D1  .X.è.Ýô..ïta?°æÑ
05E720A0  C7 BE 96 AC 62 65 D6 F3 17 C5 6A 8C 80 A6 C2 E3  Ç¾.¬beÖó.Åj..¦Âã
05E720B0  B2 46 19 05 A3 11 C2 8D 8E DE 99 A7 6F 0A 38 1D  ²F..£.Â..Þ.§o.8.
05E720C0  64 FC FD 35 90 94 32 52 FB C7 C1 AC 4C 73 2A BA  düý5..2RûÇÁ¬Ls*º
05E720D0  A6 49 8E DC C5 C1 2F 0A E7 1C AE E3 07 48 72 E0  ¦I.ÜÅÁ/.ç.®ã.Hrà
05E720E0  1E CE A3 00 C4 1D 63 74 17 E3 3E FB 39 8F 1C D3  .Î£.Ä.ct.ã>û9..Ó
05E720F0  0C 21 22 E9 CE A0 37 5E 63 C2 6E 21 D8 79 38 97  .!"éÎ 7^cÂn!Øy8.
05E72100  7A 19 D7 99 96 A0 14 6D 31 01 BF 5E 28 DF A0 9E  z.×.. .m1.¿^(ß .
05E72110  7F 8B 2B 4B 87 52 E3 50 12 24 F4 4F 03 1F 4C DD  ..+K.RãP.$ôO..LÝ
05E72120  65 09 EE 59 DD 22 B8 E4 1B 79 82 3F FA 33 1D 88  e.îYÝ"¸ä.y.?ú3..
05E72130  B1 DB B3 CC C1 AD 14 E0 3E 69 25 D4 D3 67 DC EB  ±Û³ÌÁ..à>i%ÔÓgÜë
05E72140  7C 7B C3 78 71 6F 49 9A CD FA D1 86 0F 09 6A 72  |{ÃxqoI.ÍúÑ...jr
05E72150  EC 5A 52 82 57 61 B0 9D BB E1 23 05 8A 34 1C D9  ìZR.Wa°.»á#..4.Ù
05E72160  B1 1E 66 EE B1 0B 12 38 D5 2B 30 D4 4C 66 C3 FB  ±.fî±..8Õ+0ÔLfÃû
05E72170  4E 00 60 E6 F2 48 D0 42 E0 E6 48 57 97 B0 B4 17  N.`æòHÐBàæHW.°´.
05E72180  87 A2 6E A3 F1 61 CA 0C 42 93 D5 DD 8F 4E 14 50  .¢n£ñaÊ.B.ÕÝ.N.P
05E72190  CD 76 8A 21 95 8F 87 FA 41 F8 D1 B0 3B 7B FE B4  Ív.!...úAøÑ°;{þ´
05E721A0  3C 26 EA AA 93 AA FC 90 00 41 AA 28 EE 9C 7A BA  <&êª.ªü..Aª(î.zº
05E721B0  8F 51 AF 1C F6 6C 01 E6 A9 5B 29 36 9E 1F 17 F1  .Q¯.öl.æ©[)6...ñ
05E721C0  77 51 5F 08 E7 88 9B 21 E3 F1 C6 9E B3 D0 9B 26  wQ_.ç..!ãñÆ.³Ð.&
05E721D0  71 AE E3 46 83 C3 72 A0 44 73 5B 06 9D D6 76 FC  q®ãF.Ãr Ds[..Övü
05E721E0  6E 1B A8 56 93 A9 2D D9 2F 93 0B E5 C5 93 AC E2  n.¨V.©-Ù/..åÅ.¬â
05E721F0  2D DB 85 7E 2D 70 8F 63 E8 00 4A A6 5B 86 58 26  -Û.~-p.cè.J¦[.X&
05E72200  77 AD D1 E4 24 E1 02 F3 75 27 2B 54 86 79 DB 81  w.Ñä$á.óu'+T.yÛ.
05E72210  DB 8B 3D F9 1E 05 D5 1D A4 63 E8 20 A9 52 7F F8  Û.=ù..Õ.¤cè ©R.ø
```

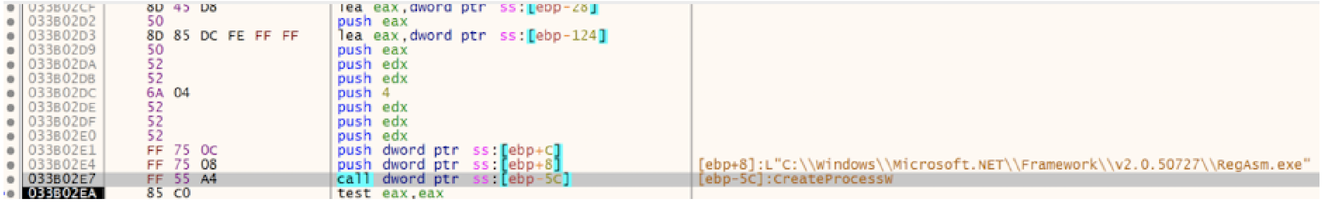
Figure 7. Encrypted stream prior to CryptDecrypt



```
te ptr [eax]=[1]=???
=1

E421A2

dress   Hex                                                ASCII
D93058  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00    MZ..........ÿÿ..
D93068  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00    ,.......@.......
D93078  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
D93088  00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00    ................
D93098  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68    ..º..´.Í!¸.LÍ!Th
D930A8  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F    is program canno
D930B8  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20    t be run in DOS 
D930C8  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00    mode....$.......
D930D8  50 45 00 00 4C 01 03 00 A1 27 E9 54 00 00 00 00    PE..L...¡'éT....
D930E8  00 00 00 00 E0 00 0E 01 0B 01 06 00 00 C8 01 00    ....à.......È..
D930F8  00 7E 01 00 00 00 00 00 92 E7 01 00 00 20 00 00    .~.......ç... ..
D93108  00 00 02 00 00 00 40 00 00 20 00 00 00 02 00 00    ......@.. ......
D93118  04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00    ................
D93128  00 A0 03 00 00 02 00 00 00 00 00 00 02 00 00 00    . ..............
D93138  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00    ................
D93148  00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00    ................
D93158  38 E7 01 00 57 00 00 00 00 20 02 00 A0 7A 01 00    8ç..W.... .. z..
D93168  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
D93178  00 00 02 00 0C 00 00 00 00 00 00 00 00 00 00 00    ................
D93188  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
D93198  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
D931A8  00 00 00 00 00 00 00 00 00 20 00 00 08 00 00 00    ......... ......
D931B8  00 00 00 00 00 00 00 00 08 20 00 00 48 00 00 00    ......... ..H...
D931C8  00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00    ........ .text...
D931D8  98 C7 01 00 00 20 00 00 00 C8 01 00 00 02 00 00    .Ç... ...È......
D931E8  00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60    ............ ..`
D931F8  2E 72 65 6C 6F 63 00 00 0C 00 00 00 00 00 02 00    .reloc..........
D93208  00 02 00 00 00 CA 01 00 00 00 00 00 00 00 00 00    .....Ê..........
D93218  00 00 00 00 40 00 00 42 2E 72 73 72 63 00 00 00    ....@..B.rsrc...
```

Figure 8. Contents decrypted after CryptDecrypt returns

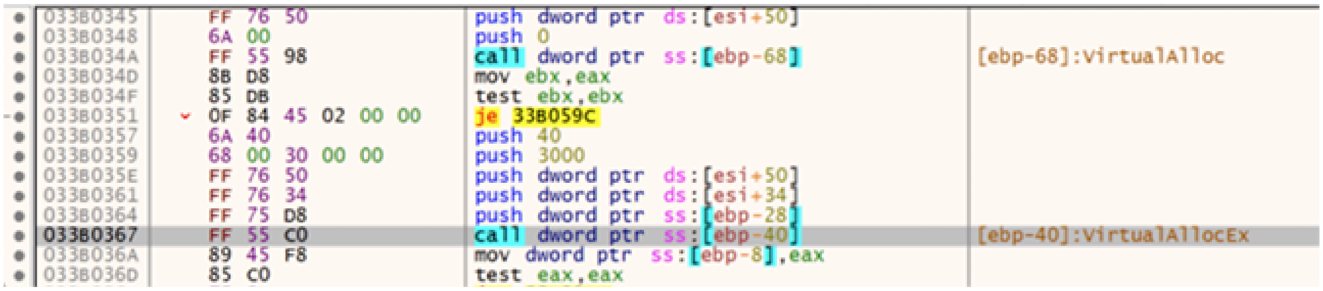
Once the contents are decrypted, it will then use the CreateProcessW function to spawn the legitimate process RegAsm.exe in a suspended state using the process creation flag `0x00000004` (`CREATE_SUSPENDED`)



```
033B02CF   8D 45 D8            lea eax,dword ptr ss:[ebp-28]
033B02D2   50                  push eax
033B02D3   8D 85 DC FE FF FF   lea eax,dword ptr ss:[ebp-124]
033B02D9   50                  push eax
033B02DA   52                  push edx
033B02DB   52                  push edx
033B02DC   6A 04               push 4
033B02DE   52                  push edx
033B02DF   52                  push edx
033B02E0   52                  push edx
033B02E1   FF 75 0C            push dword ptr ss:[ebp+C]
033B02E4   FF 75 08            push dword ptr ss:[ebp+8]     [ebp+8]:L"C:\\Windows\\Microsoft.NET\\Framework\\v2.0.50727\\RegAsm.exe"
033B02E7   FF 55 A4            call dword ptr ss:[ebp-5C]    [ebp-5C]:CreateProcessW
033B02EA   85 C0               test eax,eax
```

Figure 9. x32dbg debugger CreateProcessW function starts RegAsm.exe in suspended state

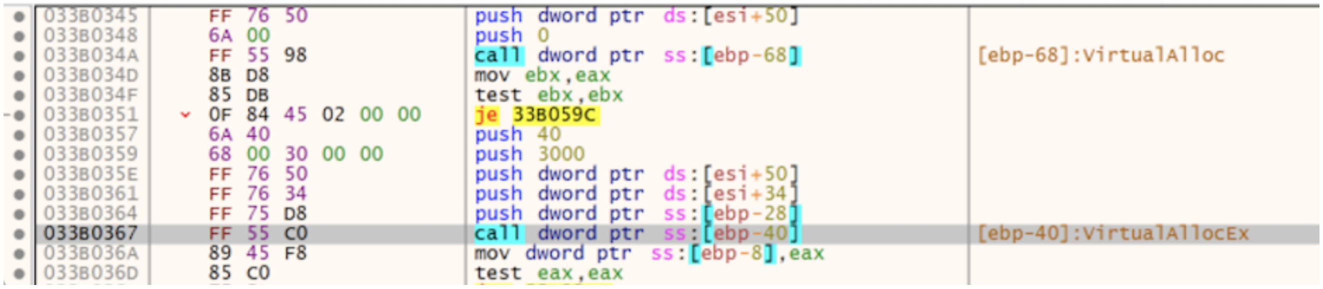
Shortly after, it proceeds to allocate memory space for the malicious payload that was decrypted earlier. This memory region is created with memory protection of `0x40` (`PAGE_EXECUTE_READWRITE`)



```
033B0345    FF 76 50             push dword ptr ds:[esi+50]
033B0348    6A 00                push 0
033B034A    FF 55 98             call dword ptr ss:[ebp-68]       [ebp-68]:VirtualAlloc
033B034D    8B D8                mov ebx,eax
033B034F    85 DB                test ebx,ebx
033B0351    0F 84 45 02 00 00    je 33B059C
033B0357    6A 40                push 40
033B0359    68 00 30 00 00       push 3000
033B035E    FF 76 50             push dword ptr ds:[esi+50]
033B0361    FF 76 34             push dword ptr ds:[esi+34]
033B0364    FF 75 D8             push dword ptr ss:[ebp-28]
033B0367    FF 55 C0             call dword ptr ss:[ebp-40]       [ebp-40]:VirtualAllocEx
033B036A    89 45 F8             mov dword ptr ss:[ebp-8],eax
033B036D    85 C0                test eax,eax
```

Figure 10. x32dbg debugger VirtualAllocEx allocating memory space

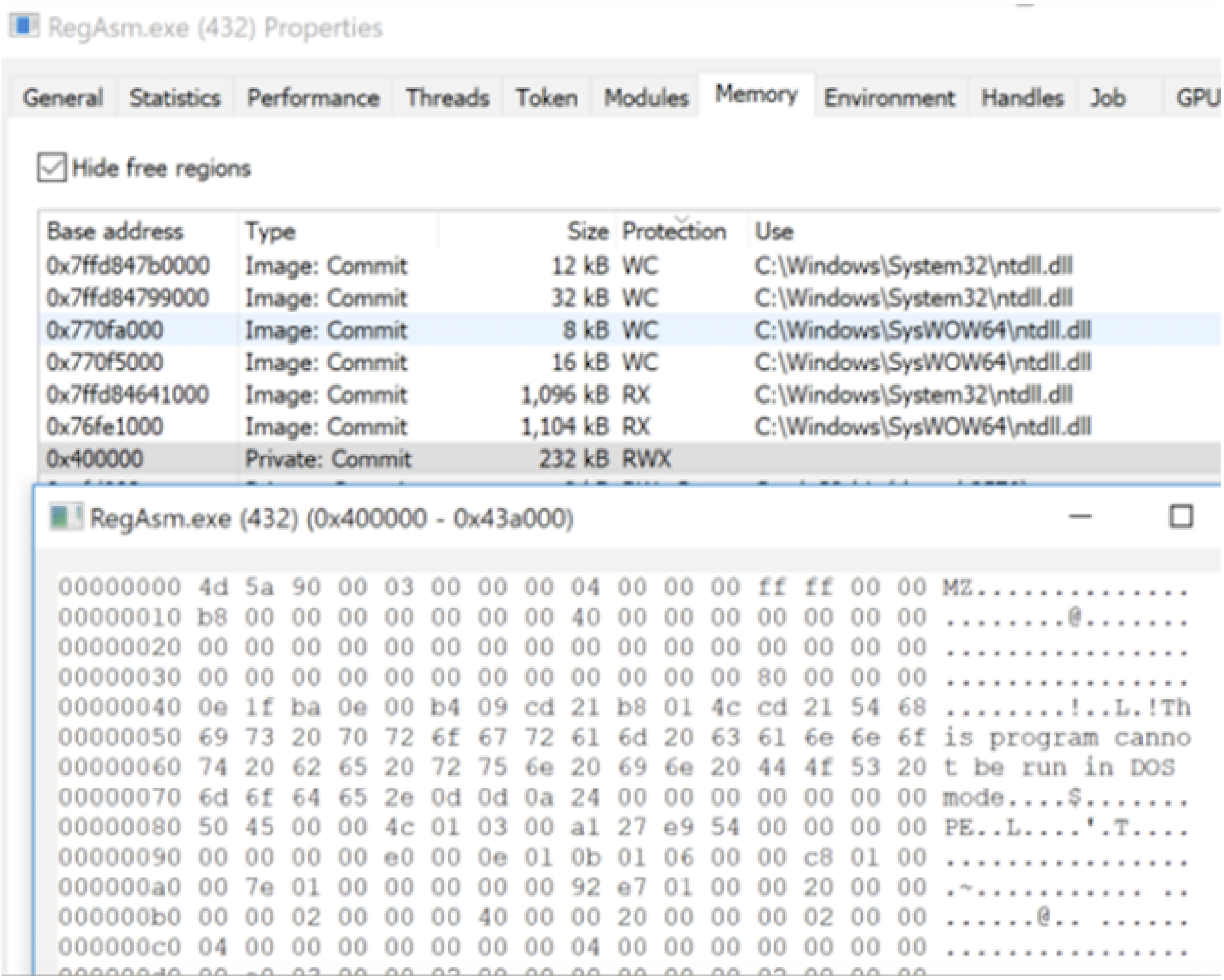
Last, the WriteProcessMemory call is seen to finally write the contents into this newly created memory region.



```
033B0345    FF 76 50             push dword ptr ds:[esi+50]
033B0348    6A 00                push 0
033B034A    FF 55 98             call dword ptr ss:[ebp-68]       [ebp-68]:VirtualAlloc
033B034D    8B D8                mov ebx,eax
033B034F    85 DB                test ebx,ebx
033B0351    0F 84 45 02 00 00    je 33B059C
033B0357    6A 40                push 40
033B0359    68 00 30 00 00       push 3000
033B035E    FF 76 50             push dword ptr ds:[esi+50]
033B0361    FF 76 34             push dword ptr ds:[esi+34]
033B0364    FF 75 D8             push dword ptr ss:[ebp-28]
033B0367    FF 55 C0             call dword ptr ss:[ebp-40]       [ebp-40]:VirtualAllocEx
033B036A    89 45 F8             mov dword ptr ss:[ebp-8],eax
033B036D    85 C0                test eax,eax
```

Figure 11. x32dbg debugger WriteProcessMemory function writing into memory region

Inspecting `RegAsm.exe` using ProcessHacker shows the memory region `0x400000` that was created earlier filled with the payload. The sample is using a well-known technique to hollow out `RegAsm.exe` and inject its payload.



☑ Hide free regions

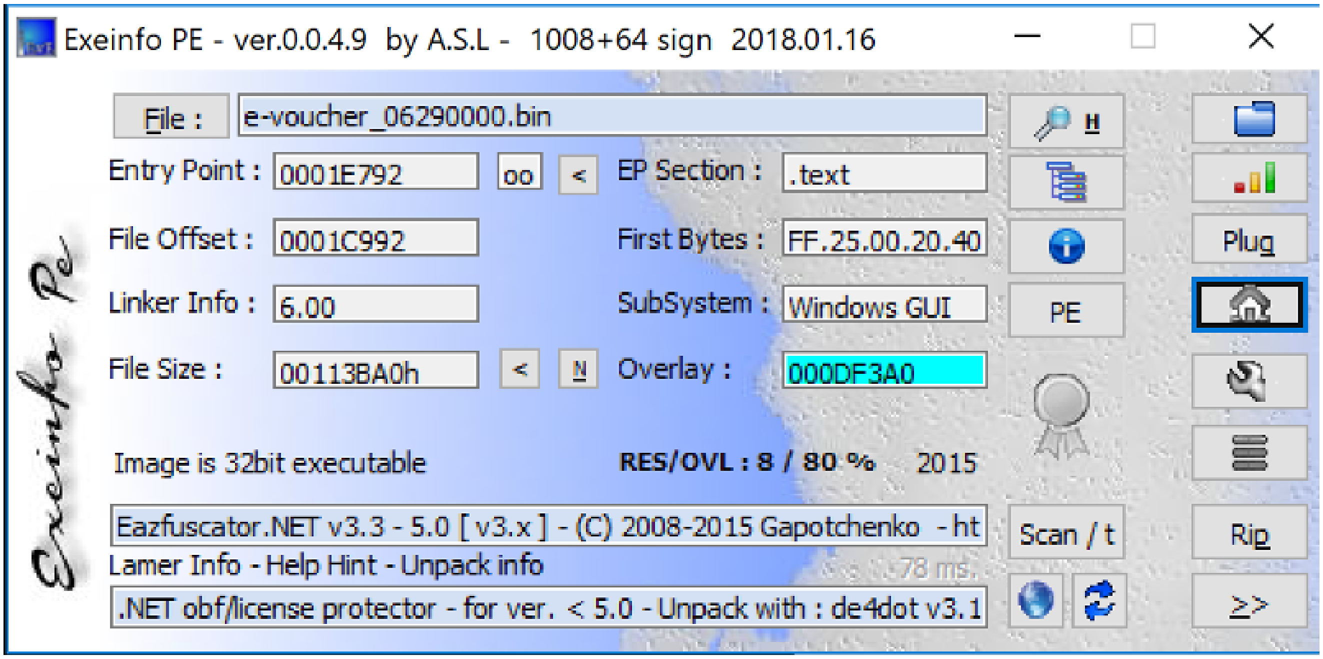
Base address	Type	Size	Protection	Use
0x7ffd847b0000	Image: Commit	12 kB	WC	C:\Windows\System32\ntdll.dll
0x7ffd84799000	Image: Commit	32 kB	WC	C:\Windows\System32\ntdll.dll
0x770fa000	Image: Commit	8 kB	WC	C:\Windows\SysWOW64\ntdll.dll
0x770f5000	Image: Commit	16 kB	WC	C:\Windows\SysWOW64\ntdll.dll
0x7ffd84641000	Image: Commit	1,096 kB	RX	C:\Windows\System32\ntdll.dll
0x76fe1000	Image: Commit	1,104 kB	RX	C:\Windows\SysWOW64\ntdll.dll
0x400000	Private: Commit	232 kB	RWX	

RegAsm.exe (432) (0x400000 - 0x43a000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS 
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
00000080 50 45 00 00 4c 01 03 00 a1 27 e9 54 00 00 00 00 PE..L....'.T....
00000090 00 00 00 00 e0 00 0e 01 0b 01 06 00 00 c8 01 00 ................
000000a0 00 7e 01 00 00 00 00 00 92 e7 01 00 00 20 00 00 .~........... ..
000000b0 00 00 02 00 00 00 40 00 00 20 00 00 00 02 00 00 ......@.. ......
000000c0 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 ................
```

Figure 12. ProcessHacker showing memory region injected with malicious code

After dumping the malicious code out of memory, we can confirm that it is a .NET built binary packed with Eazfuscator.



File : e-voucher_06290000.bin

Entry Point : 0001E792 oo < EP Section : .text

File Offset : 0001C992 First Bytes : FF.25.00.20.40

Linker Info : 6.00 SubSystem : Windows GUI

File Size : 00113BA0h < N Overlay : 000DF3A0

Image is 32bit executable RES/OVL : 8 / 80 % 2015

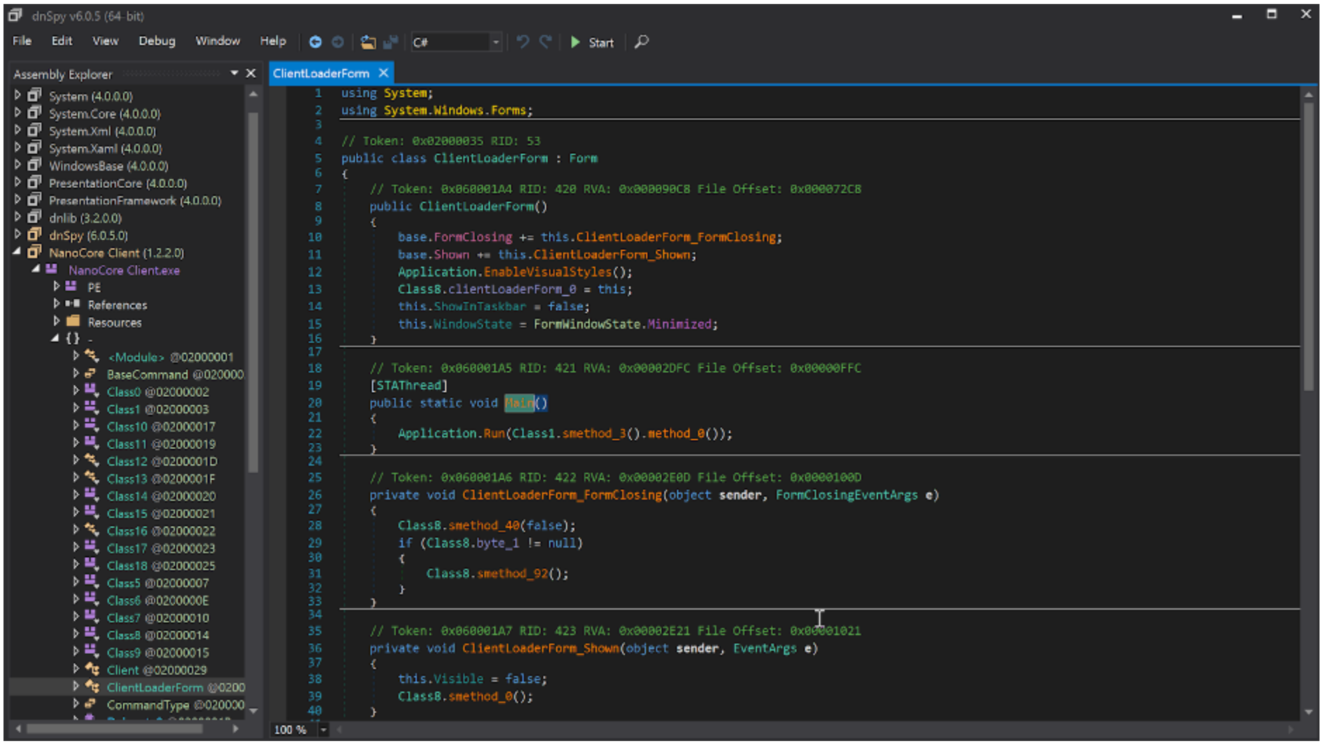
Eazfuscator.NET v3.3 - 5.0 [v3.x] - (C) 2008-2015 Gapotchenko - ht

Lamer Info - Help Hint - Unpack info 78 ms.

.NET obf/license protector - for ver. < 5.0 - Unpack with : de4dot v3.1

Figure 13. Exeinfo displaying packer information on dumped process

Running de4dot against this copy is able to deobfuscate to see readable strings.



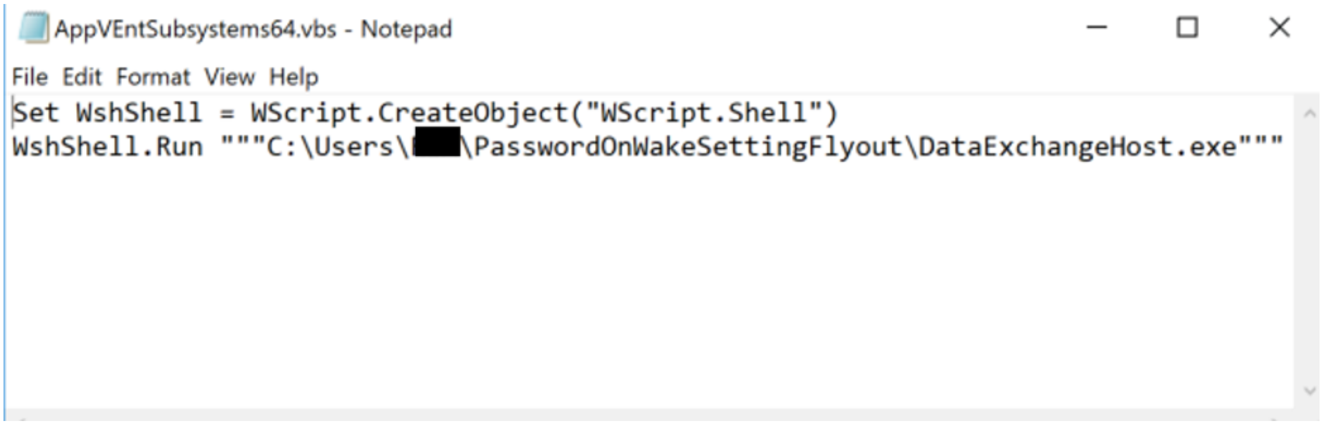
```
using System;
using System.Windows.Forms;

// Token: 0x02000035 RID: 53
public class ClientLoaderForm : Form
{
    // Token: 0x060001A4 RID: 420 RVA: 0x000090C8 File Offset: 0x000072C8
    public ClientLoaderForm()
    {
        base.FormClosing += this.ClientLoaderForm_FormClosing;
        base.Shown += this.ClientLoaderForm_Shown;
        Application.EnableVisualStyles();
        Class8.clientLoaderForm_0 = this;
        this.ShowInTaskbar = false;
        this.WindowState = FormWindowState.Minimized;
    }

    // Token: 0x060001A5 RID: 421 RVA: 0x00002DFC File Offset: 0x00000FFC
    [STAThread]
    public static void Main()
    {
        Application.Run(Class1.smethod_3().method_0());
    }

    // Token: 0x060001A6 RID: 422 RVA: 0x00002E0D File Offset: 0x0000100D
    private void ClientLoaderForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        Class8.smethod_40(false);
        if (Class8.byte_1 != null)
        {
            Class8.smethod_92();
        }
    }

    // Token: 0x060001A7 RID: 423 RVA: 0x00002E21 File Offset: 0x00001021
    private void ClientLoaderForm_Shown(object sender, EventArgs e)
    {
        this.Visible = false;
        Class8.smethod_0();
    }
```

Figure 14. DnsSpy after deobfuscation

The malware then proceeds to drop a copy of itself to the path

`C:\Users\username\PasswordOnWakeSettingFlyout\DataExchangeHost.exe`

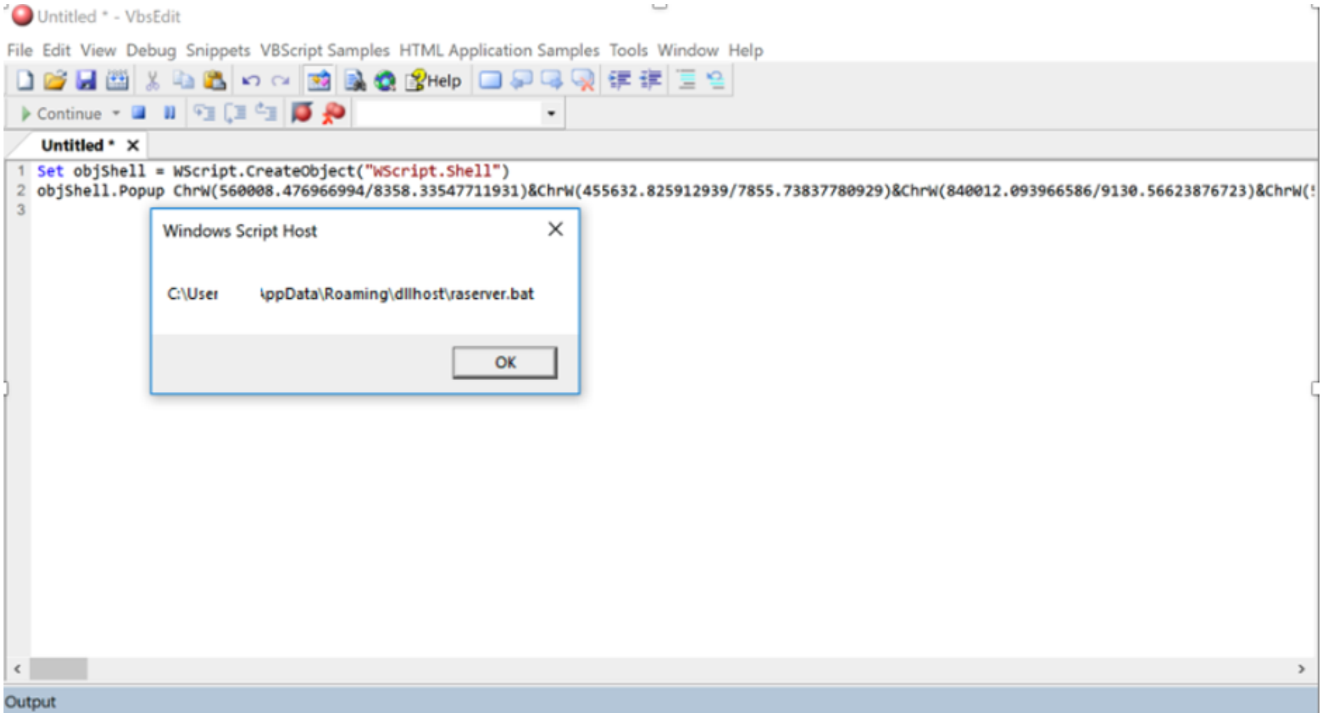
In addition, it creates persistence by using a URL shortcut in the StartUp folder that points to the copy of NanoCore RAT to survive reboot. A malicious VBS script named `AppVEntSubsystems64.vbs` is also dropped in the same directory where `DataExchangeHost.exe` resides.



```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run """C:\Users\[redacted]\PasswordOnWakeSettingFlyout\DataExchangeHost.exe"""
```

Figure 15. VBS script contents

The Falcon Complete Team has seen variations of the script above being obfuscated with the same ultimate goal such as in Figure 16.


Untitled * - VbsEdit
File Edit View Debug Snippets VBScript Samples HTML Application Samples Tools Window Help
Help
Continue
Untitled *

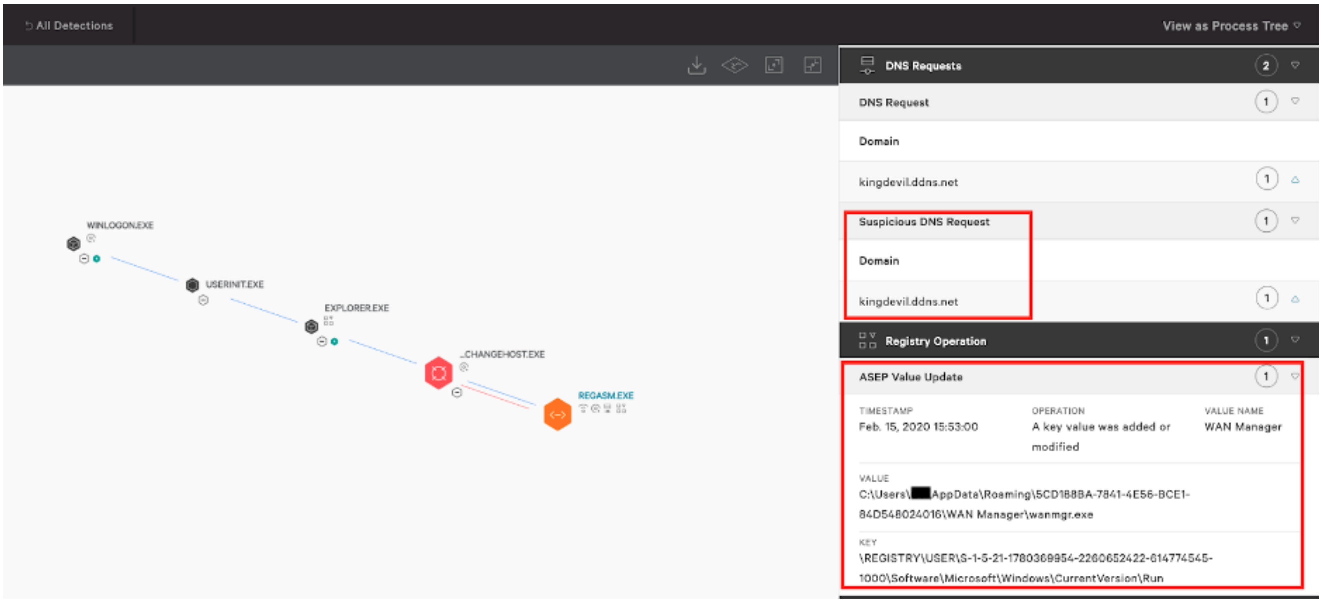
```
Set objShell = WScript.CreateObject("WScript.Shell")
objShell.Popup ChrW(560008.476966994/8358.33547711931)&ChrW(455632.825912939/7855.73837780929)&ChrW(840012.093966586/9130.56623876723)&ChrW(
```

Windows Script Host

C:\User \ppData\Roaming\dllhost\raserver.bat

Figure 16. VbsEdit debugging obfuscated script

A copy of `RegAsm.exe` is dropped onto disk and is added to the Run key to boot on user logon, as seen in Falcon’s Process Tree viewer. Falcon also logs the network connection used as the C2 in this sample, as seen in Figure 17.



DNS Requests 2

DNS Request 1

Domain

kingdevil.ddns.net 1

Suspicious DNS Request 1

Domain

kingdevil.ddns.net 1

Registry Operation 1

ASEP Value Update 1

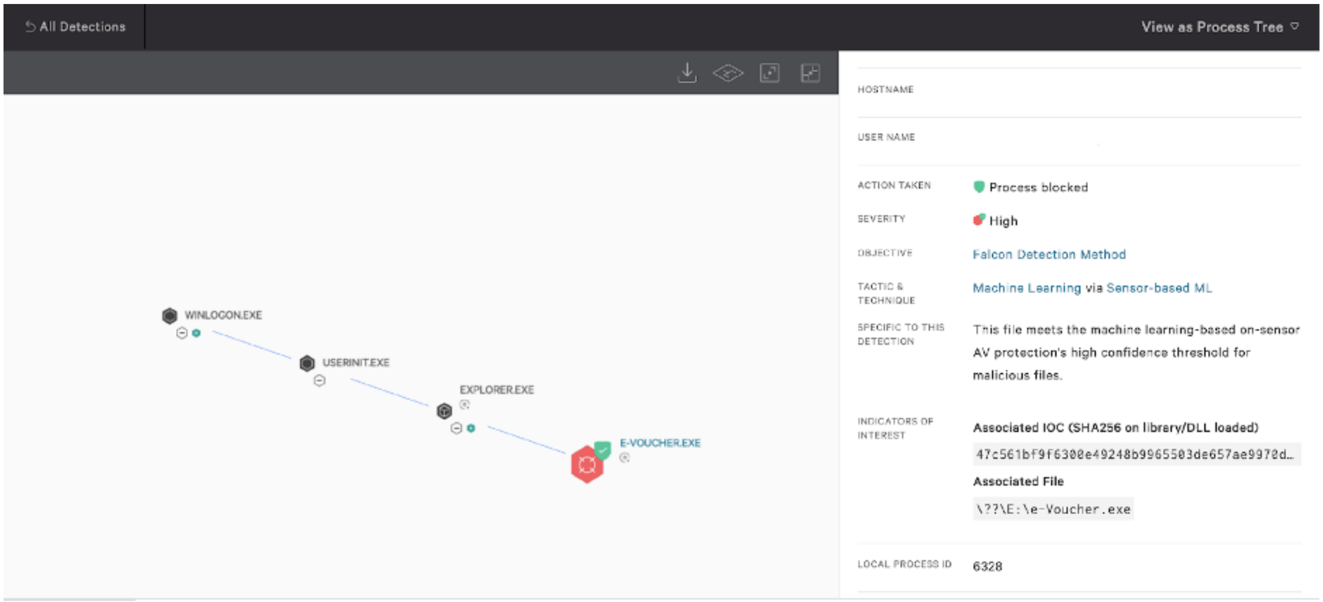
TIMESTAMP	OPERATION	VALUE NAME
Feb. 15, 2020 15:53:00	A key value was added or modified	WAN Manager

VALUE
C:\Users\[redacted]AppData\Roaming\5CD188BA-7841-4E56-BCE1-84D548024016\WAN Manager\wanmgr.exe

KEY
\REGISTRY\USER\S-1-5-21-1780369954-2260652422-614774545-1000\Software\Microsoft\Windows\CurrentVersion\Run

Figure 17. Falcon Process Tree displaying Registry Operations and DNS request

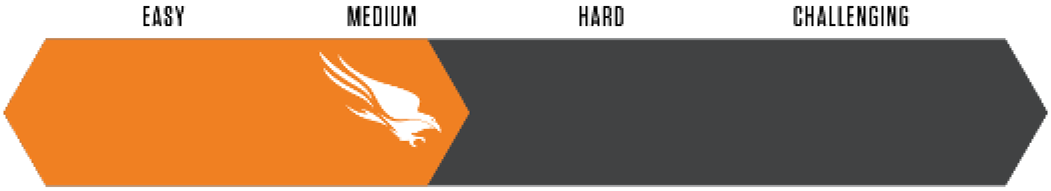
The functionality of NanoCore RAT has been covered heavily, so this blog will not focus on it. Figure 18 shows the same detection in Falcon's UI but this time being prevented after running the same sample with the detection and prevention settings set to "Aggressive."



HOSTNAME	
USER NAME	
ACTION TAKEN	Process blocked
SEVERITY	High
OBJECTIVE	Falcon Detection Method
TACTIC & TECHNIQUE	Machine Learning via Sensor-based ML
SPECIFIC TO THIS DETECTION	This file meets the machine learning-based on-sensor AV protection's high confidence threshold for malicious files.
INDICATORS OF INTEREST	Associated IOC (SHA256 on library/DLL loaded) 47c561bf9f6300e49248b9965503de657ae9970d… Associated File \??\E:\e-Voucher.exe
LOCAL PROCESS ID	6328

Figure 18. Prevention policy enabled

Remediation:



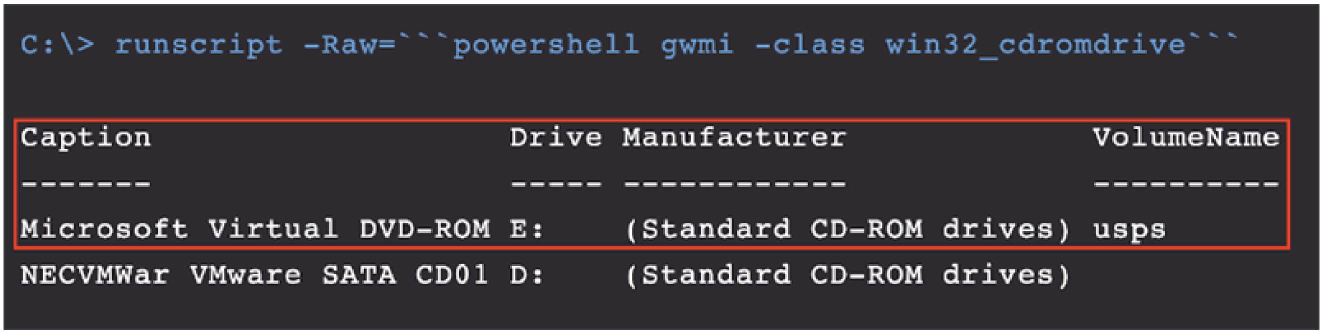
Remediation Difficulty

The remediation can be summarized in the following steps:

1. Identify and confirm detection originates from a virtual mounted drive:
   - Find the location of the disk image where it resides
   - Unmount the virtual drive
   - Remove the IMG from disk
2. Terminate the injected process
3. Remove the registry entry
4. Remove related directories and files

STEP 1: Identify and Remove the Mounted Disk Image

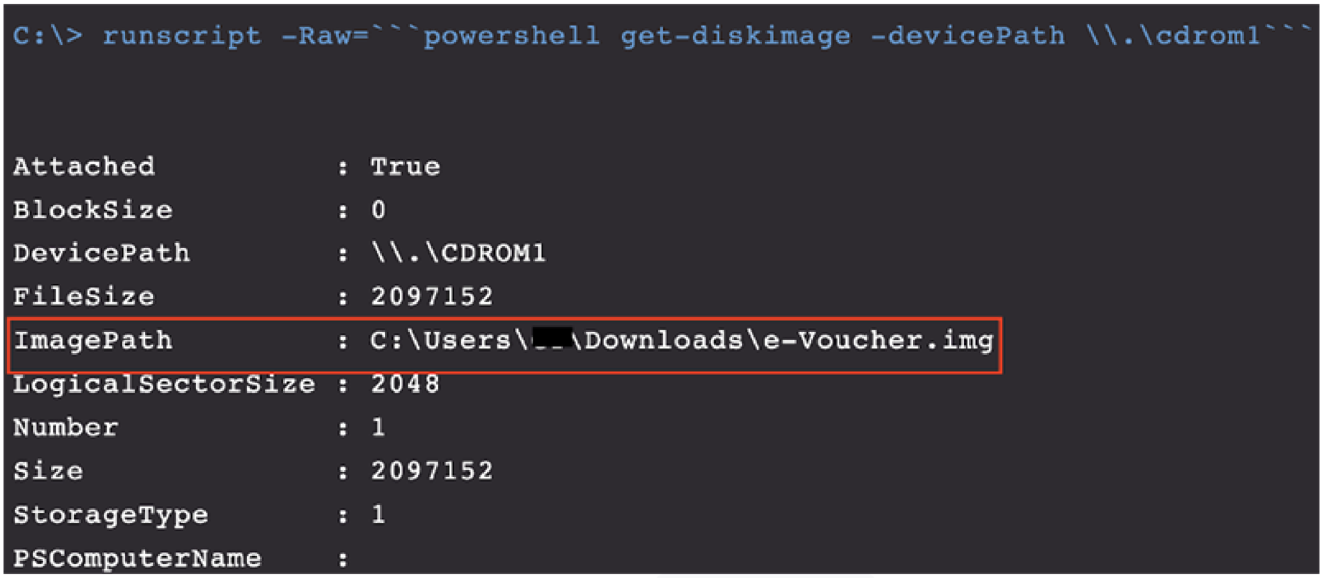
In order to identify, confirm and remove the IMG file that was mounted, we first use the class Win32_CDROMDrive from WMI in Figure 19 to provide us with information on what is currently mounted, along with the drive letter and the volume name.



```
C:\> runscript -Raw=```powershell gwmi -class win32_cdromdrive```

Caption                      Drive Manufacturer             VolumeName
-------                      ----- ------------             ----------
Microsoft Virtual DVD-ROM E:     (Standard CD-ROM drives) usps
NECVMWar VMware SATA CD01 D:     (Standard CD-ROM drives)
```

Figure 19. Output of WMI command

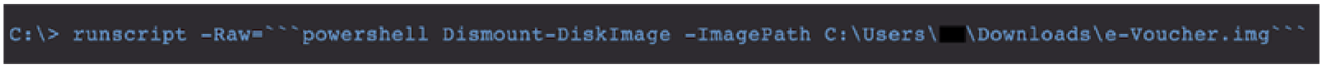
Now that we've identified what's mounted, we are using the PowerShell `Get-DiskImage` cmdlet to get the objects associated with the IMG file which will indicate where this file resides on disk.



```
C:\> runscript -Raw=```powershell get-diskimage -devicePath \\.\cdrom1```


Attached            : True
BlockSize           : 0
DevicePath          : \\.\CDROM1
FileSize            : 2097152
ImagePath           : C:\Users\[redacted]\Downloads\e-Voucher.img
LogicalSectorSize : 2048
Number              : 1
Size                : 2097152
StorageType         : 1
PSComputerName      :
```

Figure 20. Output of Powershell `Get-DiskImage` command

Use the image path obtained from the output received on the previous command to unmount this virtual disk. If the process is actively running, terminate it first. Also, you first need to unmount this disk or else you will not be able to remove it.

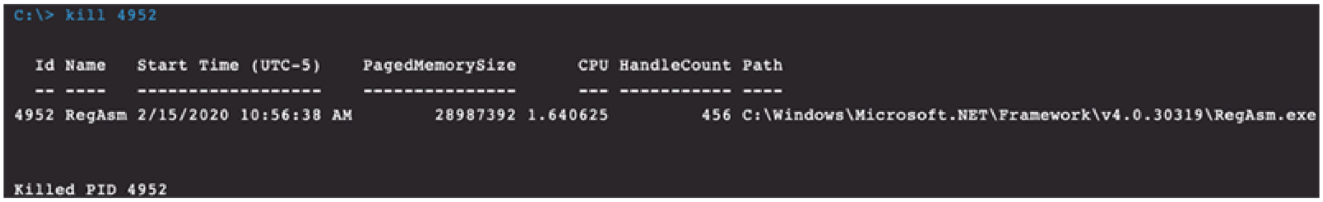


```
C:\> runscript -Raw=```powershell Dismount-DiskImage -ImagePath C:\Users\[redacted]\Downloads\e-Voucher.img```
```

Figure 21. Unmounting IMG file using Dismount-DiskImage

STEP 2: Terminate the Injected Process

From Falcon’s Process Tree, we discovered the injected RegAsm.exe process was running under the process ID 4952. Proceed to terminate this process using the built-in “kill” command using the process ID discovered.

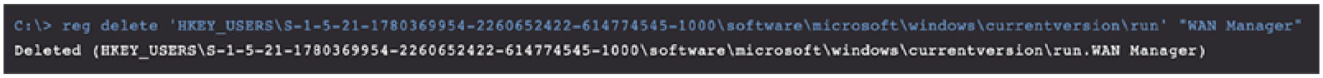


```
C:\> kill 4952

  Id Name   Start Time (UTC-5)    PagedMemorySize      CPU HandleCount Path
  -- ----   ------------------    ---------------      --- ----------- ----
4952 RegAsm 2/15/2020 10:56:38 AM        28987392 1.640625         456 C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe


Killed PID 4952
```

Figure 22. Terminated process output

STEP 3: Remove the Registry Entry

Next, we remove the registry entry that was created at infection by using the PowerShell command in Figure 23.

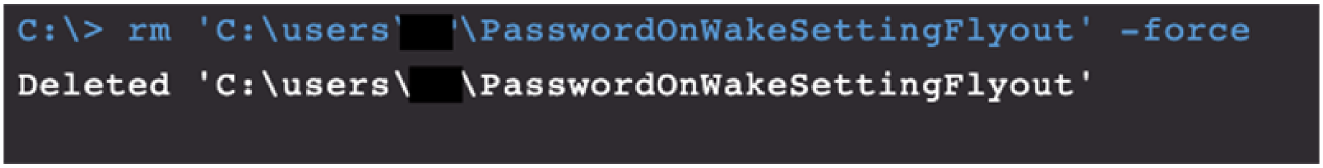


```
C:\> reg delete 'HKEY_USERS\S-1-5-21-1780369954-2260652422-614774545-1000\software\microsoft\windows\currentversion\run' "WAN Manager"
Deleted (HKEY_USERS\S-1-5-21-1780369954-2260652422-614774545-1000\software\microsoft\windows\currentversion\run.WAN Manager)
```

Figure 23. Deleting registry entry successfully

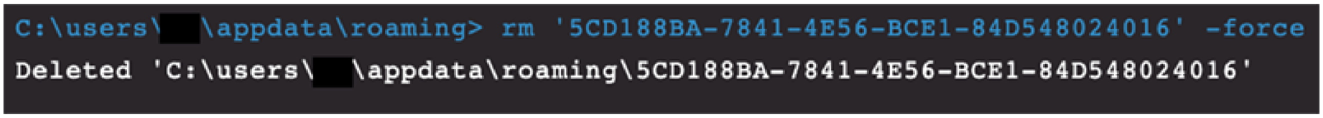
STEP 4: Remove Related Directories and Files

Last, we remove all remaining directories and files that were discovered during timeline analysis of the system.



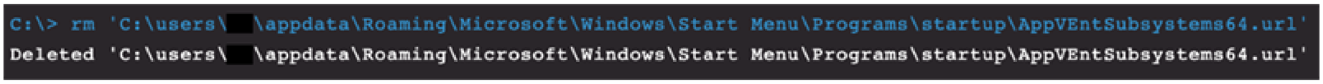
```
C:\> rm 'C:\users'[redacted]'\PasswordOnWakeSettingFlyout' -force
Deleted 'C:\users\[redacted]\PasswordOnWakeSettingFlyout'
```

Figure 24. Removing artifacts from disk output



```
C:\users\[redacted]\appdata\roaming> rm '5CD188BA-7841-4E56-BCE1-84D548024016' -force
Deleted 'C:\users\[redacted]\appdata\roaming\5CD188BA-7841-4E56-BCE1-84D548024016'
```

Figure 25. Removing artifacts from disk output



```
C:\> rm 'C:\users\[redacted]\appdata\Roaming\Microsoft\Windows\Start Menu\Programs\startup\AppVEntSubsystems64.url'
Deleted 'C:\users\[redacted]\appdata\Roaming\Microsoft\Windows\Start Menu\Programs\startup\AppVEntSubsystems64.url'
```

Figure 26. Removing artifacts from disk output

This completes the remediation steps we execute to tackle such variants when discovered. Note that in this scenario, we've purposely turned off the prevention policy while leaving the detection policy turned on for illustrative purposes.

Within the scope of our service, we've been able to observe Warzone, NanoCore and Agent Tesla RATs to be the most preferred by cybercriminals among others as seen in Figure 27.

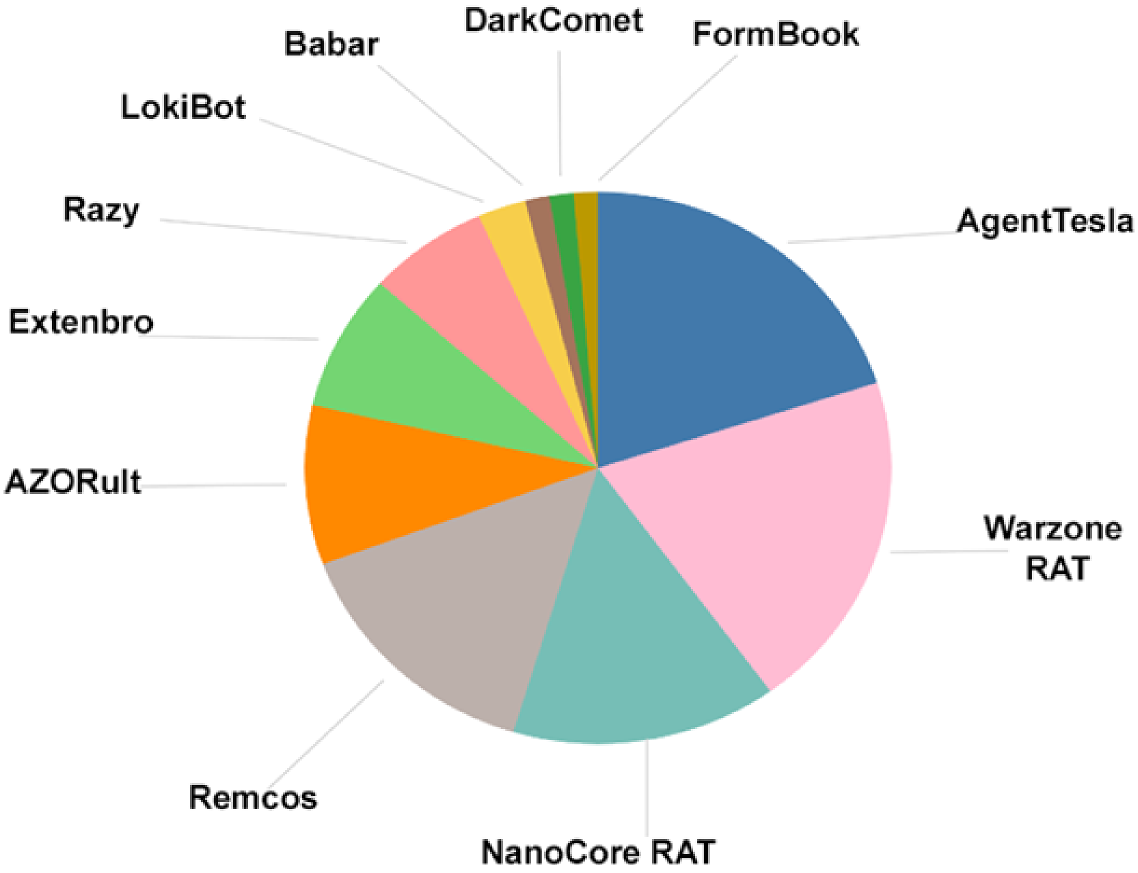


Figure 27. Malware family breakdown

The entry vector for these have primarily been phishing emails, where users download Torrent/Crack software onto their machines disguised as movies, games or music but that actually contains infected USB media.

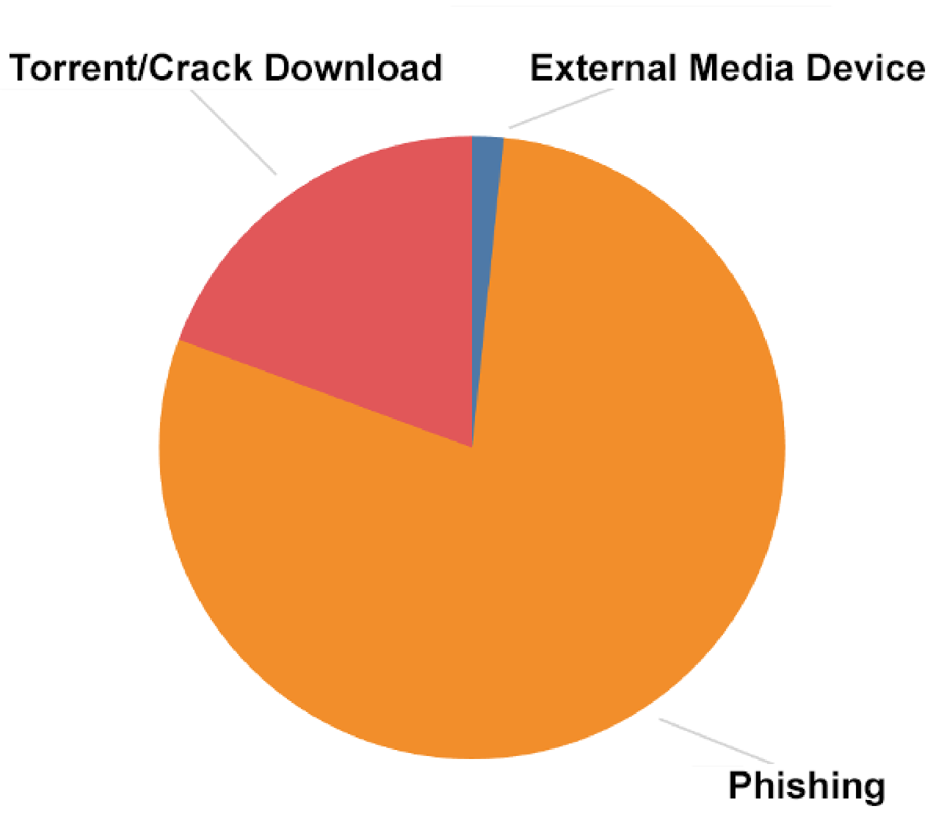


Figure 28. Entry vector breakdown

In regard to verticals, we’ve noticed these campaigns are widely spread across multiple verticals, with the hospitality sector being the most affected.

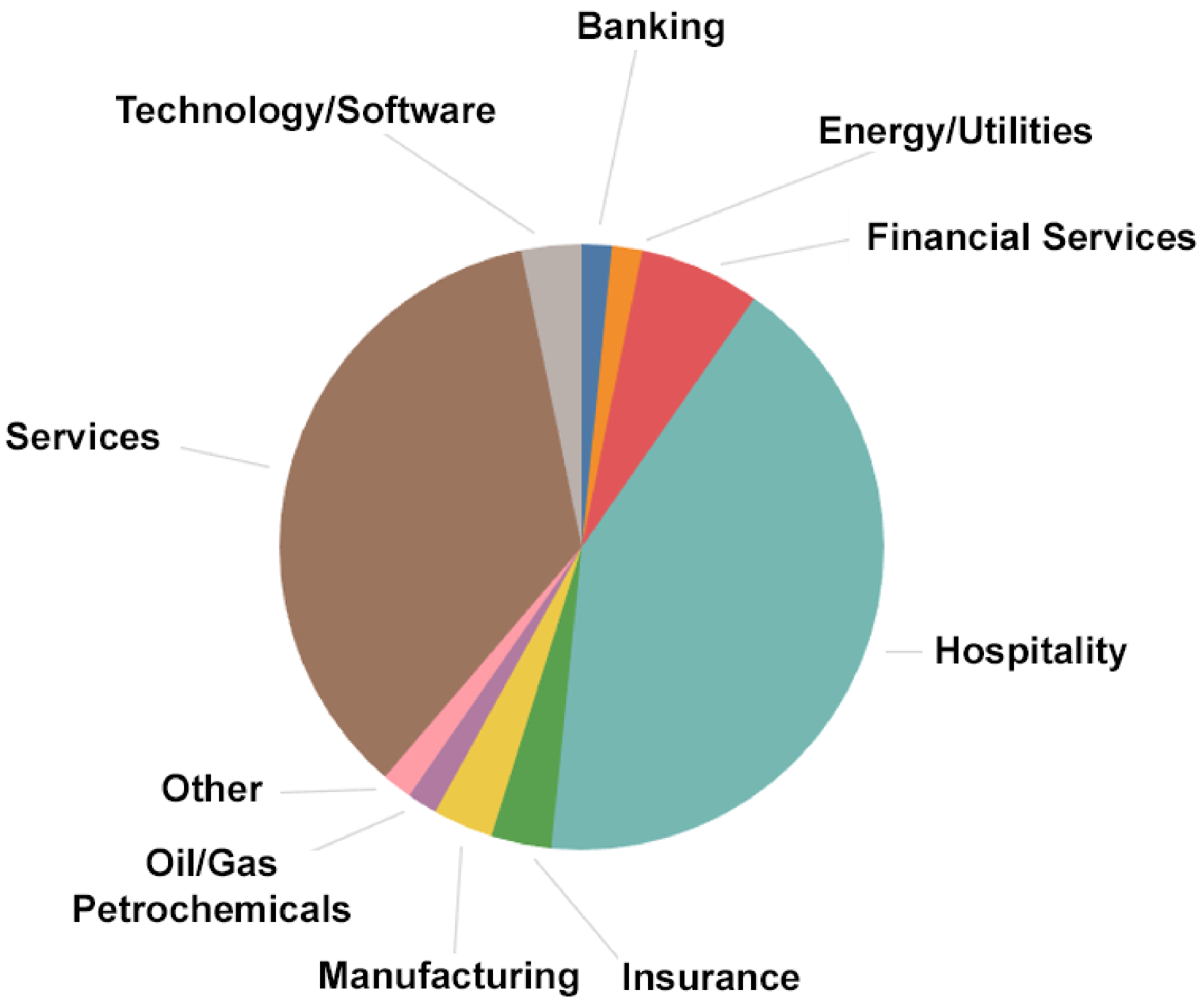


Figure 29. Affected verticals observed

Recommendations

1. Gain advanced visibility across your endpoints with an endpoint detection and response (EDR) solution such as the CrowdStrike Falcon® platform. Turn on next-gen antivirus (NGAV) preventative measures to stop malware.
2. Leverage a Layer 7 firewall that can perform deep packet inspection to examine the traffic and block P2P protocol types.
3. Observe inbound emails received during a short span of time to see the volume of disk image files being delivered as attachments. If applicable, block known disk images file types such as IMG, ISO, DAA, VHD, CDI, VMDK, etc., to reduce the attack surface.
4. Leverage a proxy to proactively block sites that are uncategorized/unknown, as we've seen new sites registered shortly before phishing campaigns are executed.
5. Incorporate a phishing awareness program internally, and routinely test employees with phishing test emails.

We’ve seen a shift toward cybercriminals using AutoIt and disk images to further achieve their objectives through various mass phishing campaigns. We believe this shift is primarily to evade detection from legacy AV software and bypass the email gateway, as most are not inspecting or blocking these file types, and no software is required to mount these disk images as Windows is able to natively mount them. We predict that in 2020, we will continue to see this trend as RATs become increasingly accessible to cybercriminals.

Additional Resources

- *Learn more about the CrowdStrike Falcon platform by visiting the webpage.*
- *Learn how you can raise your organization’s cybersecurity maturity to the highest level immediately with CrowdStrike Falcon Complete™.*
- *Learn how you can take advantage of automated malware analysis and sandbox by visiting the CrowdStrike Falcon SandboxTM webpage.*
- *Learn how CrowdStrike combines automated analysis with human intelligence to enable security teams to get ahead of the attacker’s next move by visiting the Falcon XTM webpage.*
- *Get a full-featured free trial of CrowdStrike Falcon Prevent™ and learn how true next-gen AV performs against today’s most sophisticated threats.*