Commodity .NET Packers use Embedded Images to Hide Payloads

proofpoint.com/us/blog/threat-insight/commodity-net-packers-use-embedded-images-hide-payloads

December 2, 2020

Commodity .NET Packers use Embedded Images to Hide Payloads

December 07, 2020 Proofpoint Threat Research Team

Most malware is distributed in "packed" form: typically an executable containing code to evade antivirus detection and sandboxes before extracting and executing the intended payload.

There are many commodity packers written in Microsoft .NET, usually but not always containing malware also written in .NET.

We discuss two prevalent such packers used to distribute a wide variety of malware but hiding the intended payload in images.

Steganography

Steganography is the technique of sending hidden messages in apparently innocent forms. For hiding data in images, the main techniques are:

- Store the hidden data at the end of an image file
- Store the hidden data within the image metadata (e.g., EXIF)
- Store the hidden data within the actual pixel data

To be truly "hidden" the latter would arguably mean using only the least significant bits of the data so that the image appears "normal" when rendered.

The packers discussed here generally use the entire image pixel data so aren't truly "hidden"; if they were displayed, the images would appear random.

"CyaX" packer

In this packer, the .NET executable contains a square PNG image in a .NET resource, which is typically a large proportion of the whole file size.

The image can be decoded to an intermediate executable, which contains a .NET resource which in turn can be decoded to the payload. Sometimes the intermediate executable uses an additional commodity packer such as ConfuserEx or .NET Reactor.

Details

The first stage payload is decoded from the Blue, Green, Red, and Alpha (BGRA) channels taking pixels in columns. Some versions use Red, Green, and Blue (RGB) channels instead.

For example, in sample SHA256 - 026b38e8eb0e4f505dc5601246143e7e77bbd2630b91df50622e7a14e0728675:

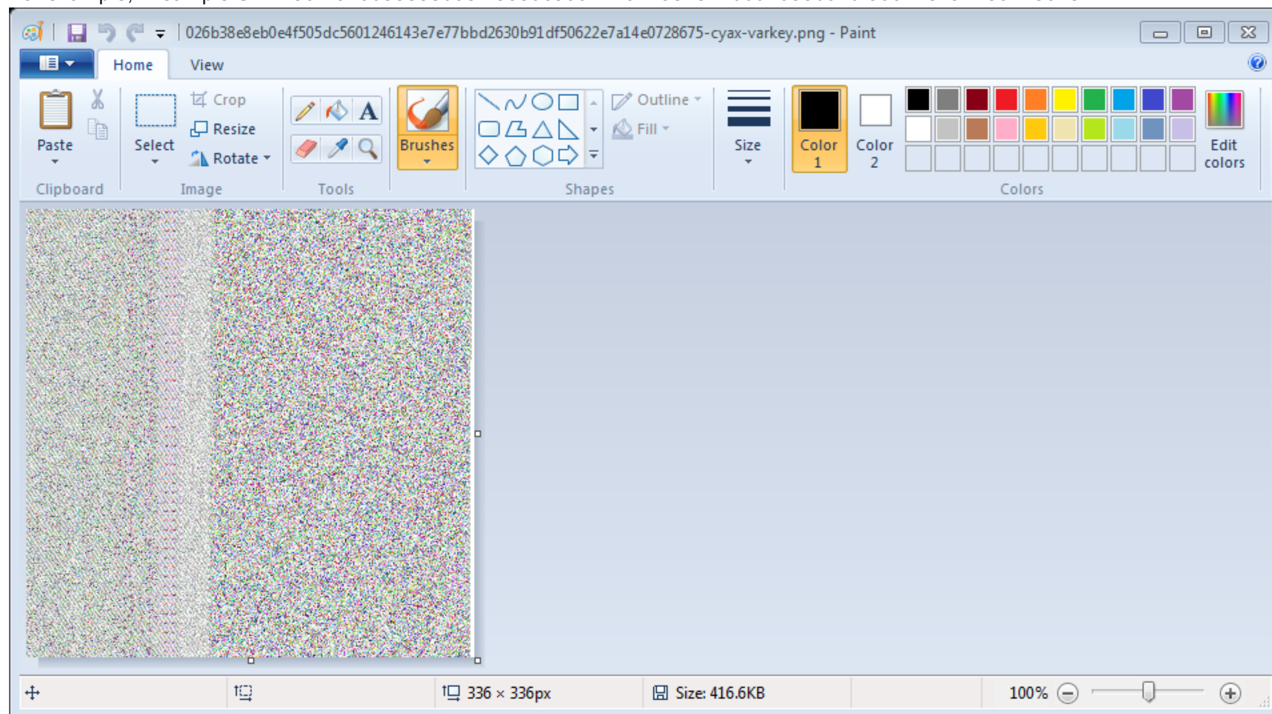

Figure 1: Image taken from sample SHA256: 026b38e8eb0e4f505dc5601246143e7e77bbd2630b91df50622e7a14e0728675

Using channels BGRA from the image we get data starting:

```
00000000  39 19 e4 17 77 02 74 23  70 74 43 74 e8 8b 02 74  |9...w.t#ptCt...t|
00000010  9b 74 74 43 74 17 74 02  34 23 74 74 43 74 17 74  |.ttCt.t.4#ttCt.t|
00000020  02 74 23 74 74 43 74 17  74 02 74 23 74 74 43 74  |.t#ttCt.t.t#ttCt|
00000030  17 74 02 74 23 74 74 43  74 17 74 02 f4 23 74 74  |.t.t#ttCt.t..#tt|
00000040  4d 6b ad 7a 02 c0 2a b9  55 fb 75 5b b9 23 20 4b  |Mk.z..*.U.u[.# K|
00000050  1d 07 63 04 65 1b 65 06  42 19 54 20 15 79 1a 6d  |..c.e.e.B.T .y.m|
00000060  00 03 16 11 63 06 62 1a  22 1d 4d 54 30 0c 27 37  |....c.b.".MT0.'7|
00000070  19 6d 10 46 5a 79 4e 7e  33 74 02 74 23 74 74 43  |.m.FZyN~3t.t#ttC|
00000080  24 52 74 02 38 22 70 74  0b 21 9e 2b 02 74 23 74  |$Rt.8"pt.!.+.t#t|
```

In general, the extracted data is then XORed with a short XOR key or the first 16 bytes of the data and possibly decompressed with gzip, yielding an intermediate stage .NET executable.

For the above sample, the XOR key is (in hex) "74 43 74 17 74 02 74 23 74", which gives the executable:

```
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 04 00  48 55 89 5f 00 00 00 00  |PE..L...HU._....|
```

This intermediate stage is often itself packed with ConfuserEx, but after unpacking that, it contains a .NET resource which contains the payload, typically XORed with two keys:, a short (often truncated Unicode) one, followed by a 16-byte key stored at the start of the resulting file.

In the above sample, the intermediate executable is packed with .NET Reactor. After deobfuscation with a tool such as de4dot, the deobfuscated executable contains a resource "2EJp1.resources" which starts:

```
00000000  a8 48 3d cd ca 1e b7 b9  97 05 85 6f 72 c0 4d 7b  |.H=........or.M{|
00000010  e5 30 ad f4 c9 2b b7 91  93 0d 85 58 8d 06 4d 59  |.0...+.....X..MY|
00000020  10 53 3d c1 ca 03 b7 99  d7 3a 85 61 72 db 4d 60  |.S=......:.ar.M`|
00000030  a8 66 3d e9 ca 0b b7 ae  97 03 85 43 72 e2 4d 55  |.f=........Cr.MU|
00000040  a8 4e 3d e1 ca 3c b7 97  97 21 85 7a f2 d7 4d 7d  |.N=..<...!.z..M}|
00000050  a6 59 87 d8 ca b1 be 78  b6 a0 84 03 bf de 19 1d  |.Y.....x........|
00000060  c1 02 1d 9f b8 48 d0 fe  f6 40 a5 04 13 99 23 2d  |.....H...@....#-|
00000070  dc 68 5f a8 ea 6c c2 d7  b7 6c eb 4f 36 8f 1e 5b  |.h_..l...l.O6..[|
00000080  c5 05 59 91 e4 26 ba 9b  b3 0d 85 58 72 f9 4d 59  |..Y..&.....Xr.MY|
```

XORing with key "00 77 00 55 00 6c 00 59 00 71 00 79 00 4e" ("wUlYqyNZJIbjVN" in Unicode, truncated to half the length):

```
00000000  a8 3f 3d 98 ca 72 b7 e0  97 74 85 16 72 8e 4d 0c  |.?=..r...t..r.M.|
00000010  e5 65 ad 98 c9 72 b7 e0  93 74 85 16 8d 71 4d 0c  |.e...r...t...qM.|
00000020  10 3f 3d 98 ca 72 b7 e0  d7 74 85 16 72 8e 4d 0c  |.?=..r...t..r.M.|
00000030  a8 3f 3d 98 ca 72 b7 e0  97 74 85 16 72 8e 4d 0c  |.?=..r...t..r.M.|
00000040  a8 3f 3d 98 ca 72 b7 e0  97 74 85 16 f2 8e 4d 0c  |.?=..r...t....M.|
00000050  a6 20 87 96 ca c6 be 2d  b6 cc 84 5a bf af 19 64  |. .....-...Z...d|
00000060  c1 4c 1d e8 b8 1d d0 92  f6 19 a5 75 13 e0 23 63  |.L.........u..#c|
00000070  dc 1f 5f fd ea 00 c2 8e  b7 1d eb 36 36 c1 1e 2c  |.._........66..,|
00000080  c5 50 59 fd e4 7f ba ea  b3 74 85 16 72 8e 4d 0c  |.PY......t..r.M.|
```

and then XORing with the first 16 bytes of the result gives the payload, Agent Tesla (a prevalent information stealer) in this case:

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000020  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00  |................|
00000050  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000060  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000070  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000080  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
```

In some early versions of this packer, this .NET resource was named "CyaX_Sharp.Properties.Resources.resources" hence the name we have given to this packer family.

Gzip variant

As mentioned above, some samples use the Red, Green, and Blue (RGB) channels, and some compress the intermediate executable with gzip.

For example, in sample SHA256 - 083521fa4522245adc968b1b7dd18da29b193fd41572114c9d7dd927918234ea:

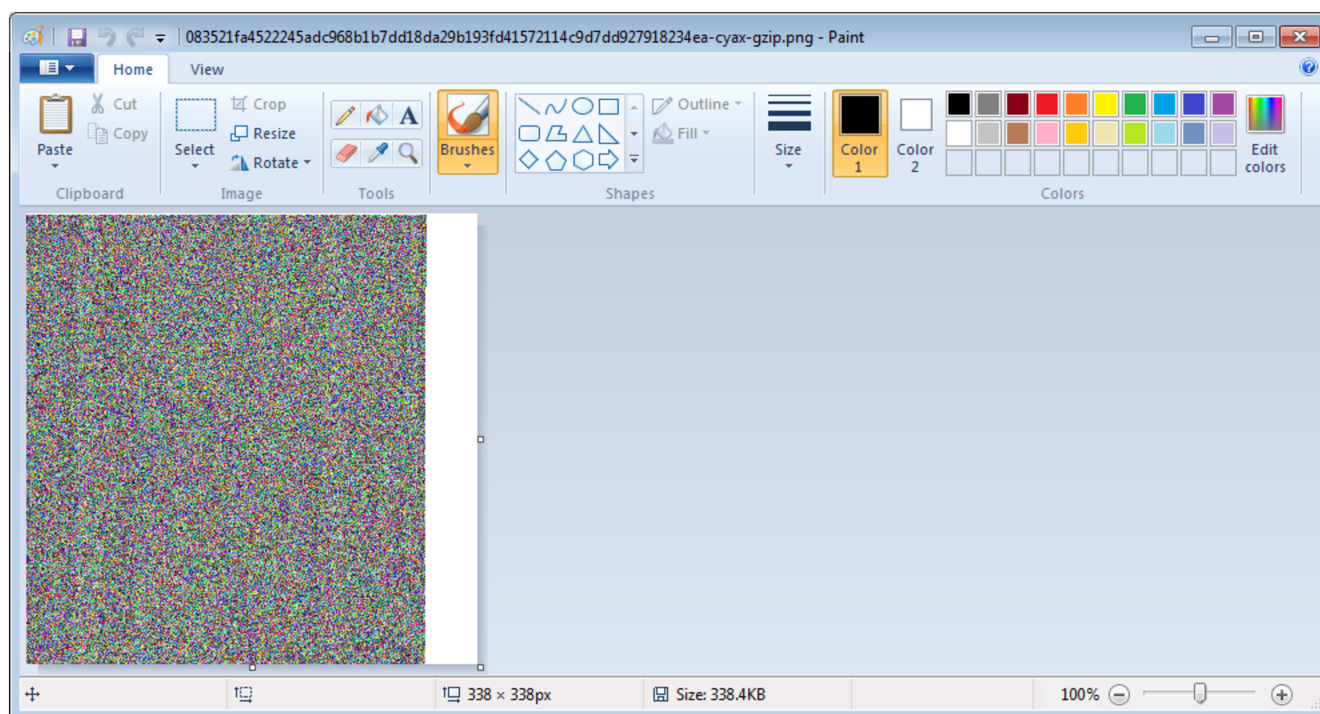

Figure 2: Image taken from sample SHA256: 083521fa4522245adc968b1b7dd18da29b193fd41572114c9d7dd927918234ea

the image uses RGB channels which decode to:

```
00000000  24 54 35 24 1b b8 2c 04  33 24 04 33 20 04 a7 59  |$T5$..,.3$.3 ..Y|
00000010  03 af 70 41 c1 db df ea  fd 8d 5c 52 73 45 52 63  |..pA......\RsERc|
00000020  c0 52 87 b1 e7 3a a1 44  6c a1 26 8e 2b 25 c5 03  |.R...:.Dl.&.+%..|
00000030  a7 0e bd 96 af 2b 7c 60  22 74 10 02 ba e3 3a ae  |.....+|`"t....:.|
00000040  f5 f7 5c e2 2f 6a cb e0  17 98 0a b9 9a 58 26 41  |..\./j.......X&A|
00000050  3c 25 15 27 37 2e 0a c8  fa d8 7f d9 5c 42 72 25  |<%.'7.......\Br%|
00000060  ca 7a 4c 1a f0 7d 9b 7a  2e 8f af 98 8f af 99 cf  |.zL..}.z........|
00000070  19 06 d5 5e 98 f4 b6 1f  0b 81 bd 2a cf 49 ea d6  |...^.......*.I..|
00000080  cc a2 5f cc df db 4f 2e  c1 d1 3b c1 e5 17 c5 3c  |.._...O...;....<|
```

XORing with key (in hex) "24 04 33" gives:

```
00000000  00 50 06 00 1f 8b 08 00  00 00 00 00 04 00 94 7d  |.P.............}|
00000010  07 9c 54 45 f2 ff db d9  d9 89 6f 76 77 76 76 67  |..TE......ovwvvg|
00000020  f3 76 83 82 c3 3e 92 60  68 92 02 8a 18 01 c1 30  |.v...>.`h......0|
00000030  83 0a 8e b2 ab 18 58 64  11 50 14 31 9e e7 09 8a  |......Xd.P.1....|
00000040  f1 c4 78 e6 1c 4e cf d3  33 9c 39 9d 9e 6b 02 45  |..x..N..3.9..k.E|
00000050  0f 01 11 14 13 2a 39 ec  fe eb 5b dd 6f 66 76 16  |.....*9...[.ofv.|
00000060  ee 7e 7f 3e f4 4e bf 7e  1d ab ab ab ab ab aa eb  |.~.>.N.~........|
00000070  1d 35 f1 5a ab d0 b2 2c  2f 85 8e 0e cb 7a ce d2  |.5.Z...,/....z..|
00000080  ff 86 5b ff fb df 7c 0a  c5 e2 1f c5 d6 33 c1 0f  |..[...|......3..|
```

which is a 4-byte DWORD containing the uncompressed file size, followed by a gzip-ed file, starting with a 10-byte gzip header, which decompresses to the intermediate .NET executable:

```
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  30 5a 8e 5f 00 00 00 00  |PE..L...0Z._....|
```

This contains a .NET resource "d2o6x4FhIdl.resources" starting:

```
00000000  07 48 8c 96 50 50 1b 88  be dc 38 28 0f 4b eb ca  |.H..PP....8(.K..|
00000010  09 74 6f d5 38 23 79 e3  dd be 38 29 f0 f7 8d b9  |.to.8#y...8)....|
00000020  bf 45 8c b7 50 44 1b e3  98 be 7b 4f 7c 4b e6 ca  |.E..PD....{O|K..|
00000030  65 2e eb d5 50 45 1b a0  be cd 38 24 0f 29 8d ad  |e...PE....8$.)..|
00000040  07 2e ea d5 13 23 68 e3  d5 be 5a 4f e8 4b 8d ac  |.....#h...ZO.K..|
00000050  09 72 36 a8 50 fc 12 4c  9f 61 39 03 a4 6a 9a a2  |.r6.P..L.a9..j..|
00000060  1d 5d c7 a5 40 4c 1b 91  df b5 18 6f 6e 56 e3 ce  |.]..@L.....onV..|
00000070  73 6c ee d7 70 51 08 8d  dd d7 25 6f 20 04 bc ea  |sl..pQ....%o ...|
00000080  0d 41 e8 d6 7e 6d 16 9a  9a d5 38 2d 0f 2c 8d ca  |.A..~m....8-.,..|
```

which when XORed with keys "00 66 00 43 00 73 00 6b 00 62 00 67 00" ("fCskbgkLbLArI" in Unicode, truncated) and then "07 2e 8c d5 50 23 1b e3 be be 38 4f 0f 4b 8d ca" gives:

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000020  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00  |................|
00000050  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000060  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000070  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000080  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
```

which contains the payload, Agent Tesla again.

Steganographic variant

In a recent variation of this packer, the first stage payload is actually stored in a second PNG image extracted from the least significant bits of the Red, Green, and Blue channels in the first image, taking pixels in rows (so "proper" steganography in this case). The intermediate stage .NET executable is then extracted from the Blue, Green, Red, and Alpha channels of the second image with pixels taken in columns, without XOR this time.

For example, in sample SHA256 – 04794ec7e7eb5c6611aada660fb1716a91e01503fb4703c7d2f2099c089c9017:

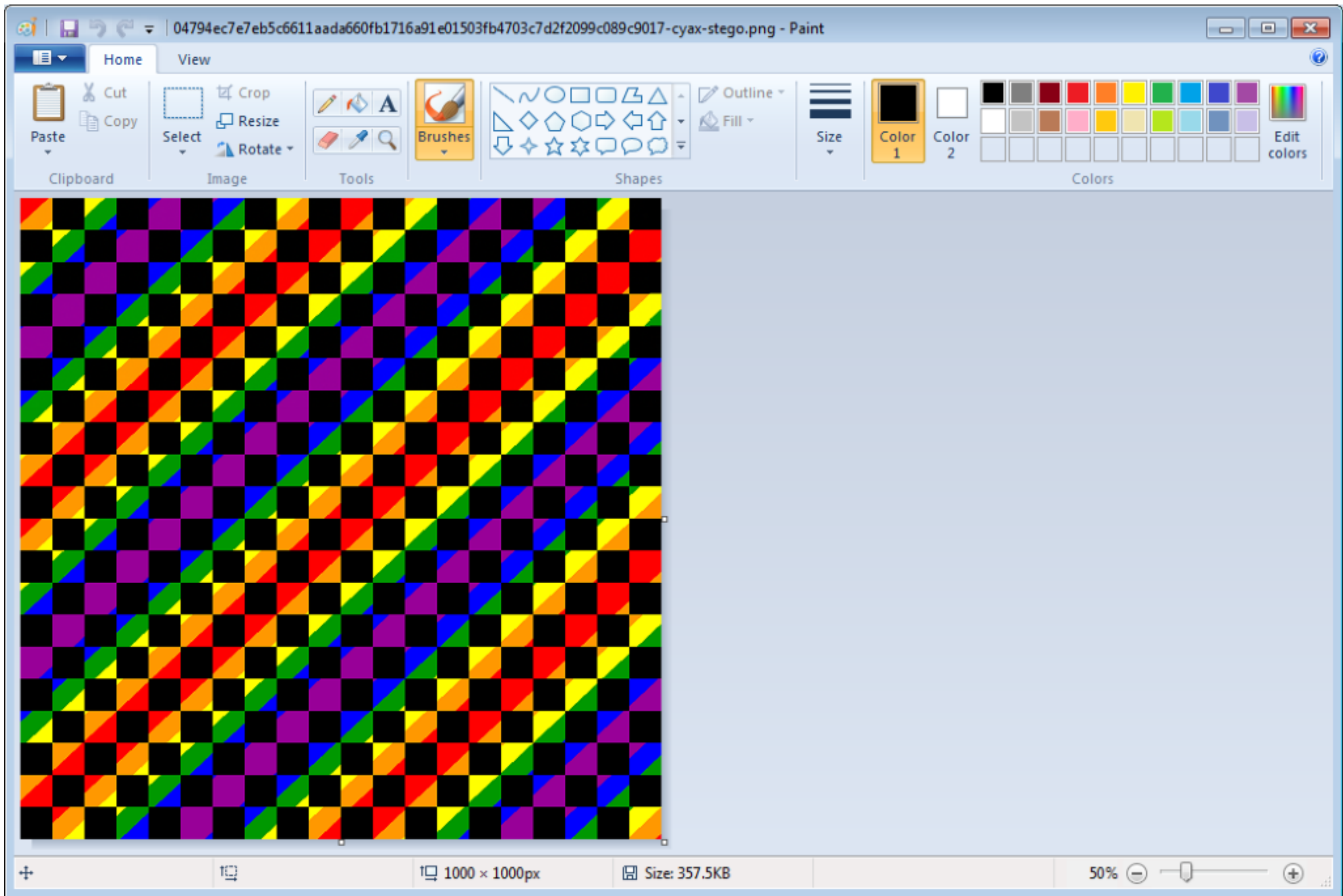

Figure 3: Image taken from sample SHA256: 04794ec7e7eb5c6611aada660fb1716a91e01503fb4703c7d2f2099c089c9017

the image has RGB channels and, taking pixels by rows first rather than columns, leads to:

```
00000000  ff 01 00 ff 01 00 fe 01  00 fe 00 00 fe 01 01 ff  |................|
00000010  01 00 fe 00 00 fe 00 00  fe 00 00 fe 00 00 fe 00  |................|
00000020  01 fe 00 01 fe 00 00 ff  00 00 fe 00 01 fe 01 00  |................|
00000030  fe 01 01 ff 00 00 ff 00  01 ff 01 00 fe 00 01 fe  |................|
00000040  01 00 ff 01 00 fe 00 00  fe 01 00 ff 00 00 fe 00  |................|
00000050  00 ff 00 01 ff 00 00 fe  00 01 fe 01 00 fe 00 00  |................|
00000060  fe 00 00 fe 00 00 fe 00  00 fe 00 00 fe 00 00 fe  |................|
00000070  00 00 fe 00 00 fe 00 00  ff 00 01 ff 00 00 fe 00  |................|
00000080  01 fe 00 01 fe 98 01 fe  98 00 fe 99 00 fe 99 00  |................|
```

(There is also an Alpha channel, with all values set to 0xff.)

Taking groups of 8 bytes and then the least significant bits in reverse order gives us (for example "ff 01 00 ff 01 00 fe 01" -> "10011011" -> 0x9b):

```
00000000  9b e0 01 00 89 50 4e 47  0d 0a 1a 0a 00 00 00 0d  |.....PNG........|
00000010  49 48 44 52 00 00 00 c6  00 00 00 c6 08 06 00 00  |IHDR............|
00000020  00 89 9b ff 5d 00 00 00  01 73 52 47 42 00 ae ce  |....]....sRGB...|
00000030  1c e9 00 00 00 04 67 41  4d 41 00 00 b1 8f 0b fc  |......gAMA......|
00000040  61 05 00 00 00 09 70 48  59 73 00 00 0e c3 00 00  |a.....pHYs......|
00000050  0e c3 01 c7 6f a8 64 00  00 ff a5 49 44 41 54 78  |....o.d....IDATx|
00000060  5e 8c fd 0b 5c 54 d5 fa  07 8c 3f c3 45 2e 72 55  |^...\T....?.E.rU|
00000070  47 31 51 51 86 8b a0 36  c3 30 30 c0 00 b3 85 51  |G1QQ...6.00....Q|
00000080  4c d0 6d 0c 20 de 47 66  8f 30 0c 97 66 f6 28 94  |L.m. .Gf.0..f.(.|
```

This is a file size stored in a DWORD (0x1e09b) followed by the second PNG image. Using BGRA and columns first, this decodes to:

```
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  0b 84 8c 5f 00 00 00 00  |PE..L......_....|
```

which contains a .NET resource "biGzxmYEphCl":

```
00000000  01 5a fb 00 77 00 79 00  50 00 65 00 99 ff 65 00  |.Z..w.y.P.e...e.|
00000010  f4 00 6b 00 74 00 79 00  14 00 65 00 66 00 65 00  |..k.t.y...e.f.e.|
00000020  4c 00 6b 00 74 00 79 00  54 00 65 00 66 00 65 00  |L.k.t.y.T.e.f.e.|
00000030  4c 00 6b 00 74 00 79 00  54 00 65 00 66 01 65 00  |L.k.t.y.T.e.f.e.|
00000040  42 1f d1 0e 74 b4 70 cd  75 b8 64 4c ab 21 31 68  |B...t.p.u.dL.!1h|
00000050  25 73 4b 70 06 6f 1e 72  35 6d 45 63 07 6e 0b 6f  |%sKp.o.r5mEc.n.o|
00000060  38 20 09 65 54 72 0c 6e  74 69 0b 20 22 4f 36 20  |8 .eTr.nti. "O6 |
00000070  21 6f 0f 65 5a 0d 74 0a  70 00 65 00 66 00 65 00  |!o.eZ.t.p.e.f.e.|
00000080  28 7c ec f9 54 1d 90 aa  74 1d 8c aa 46 1d 8c aa  |(|..T...t...F...|
```

which when XORed with "4c 00 6b 00 74 00 79 00 54 00 65 00 66 00 65 00" ("LktyTefe" in Unicode):

```
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 00 01 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  64 7c 87 f9 20 1d e9 aa  20 1d e9 aa 20 1d e9 aa  |d|.. ... ... ...|
```

gives the payload, which in this case is Remcos RAT.

"Hectobmp" packer

In this packer, the .NET executable contains typically several hundred small images in .NET resources, which each contain a part of the payload and need to be reassembled in the correct order.

Earlier versions used the BMP file format, and later versions have switched to using PNG. The name we have given to this packer comes from "hecto-" from the metric system prefix for a hundred.

Details

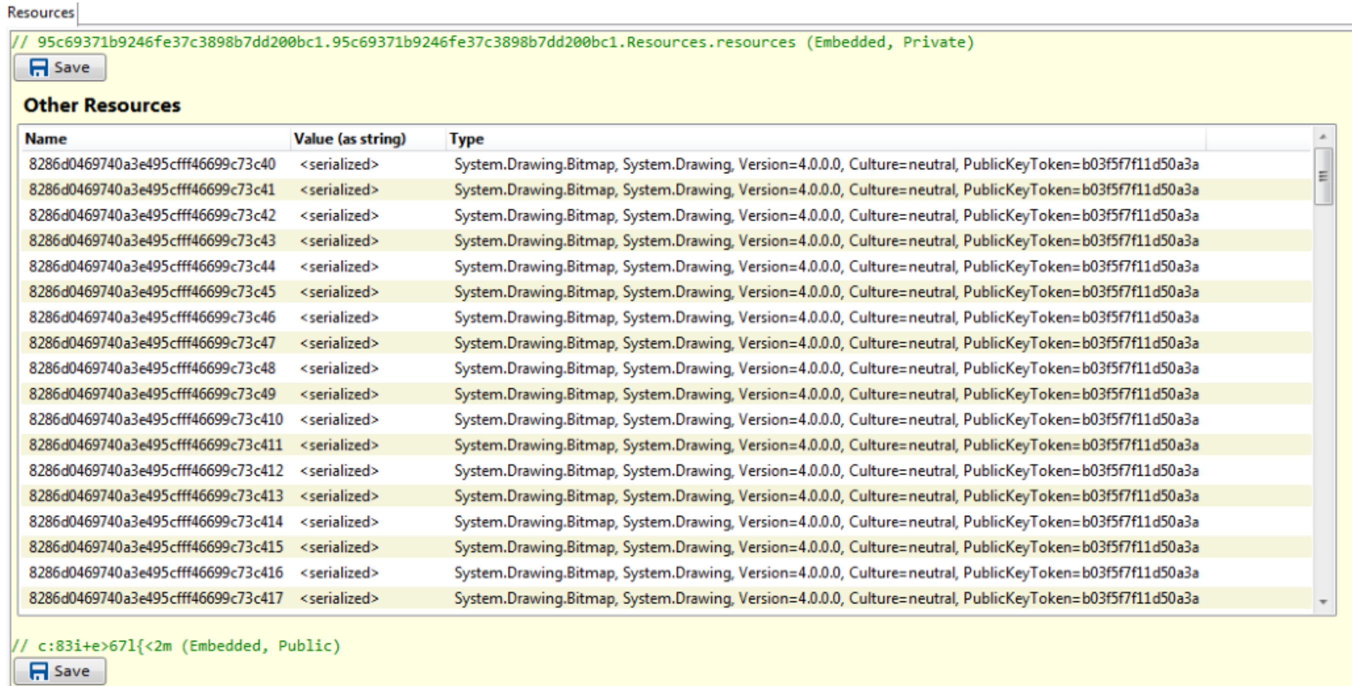

Figure 4: .NET resources list (from ILSpy)

For example, in sample SHA256 – 0091c6bdceecf3e0143b4eaaefca1cd56cbfdfc55f99c167f9dd1f3a48928bb5:

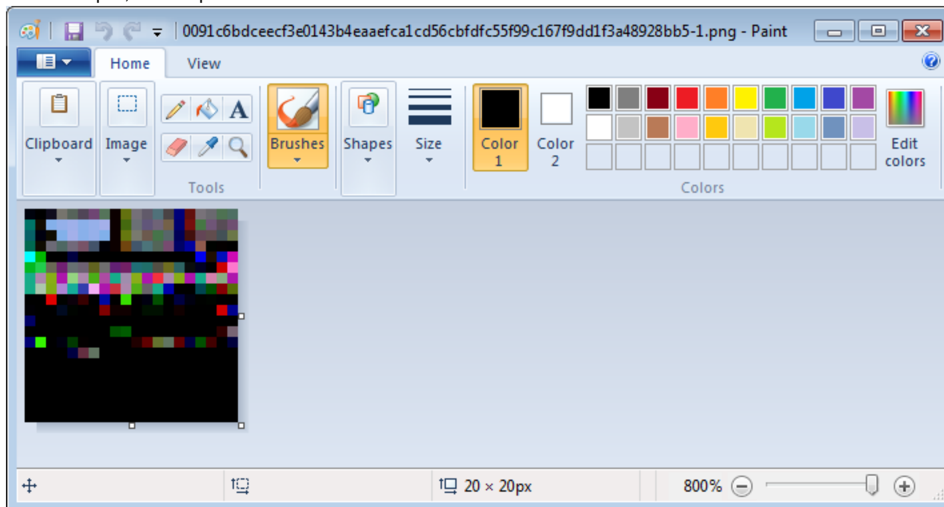

Figure 5: First image taken from sample SHA256: 0091c6bdceecf3e0143b4eaaefca1cd56cbfdfc55f99c167f9dd1f3a48928bb5

which contains 135 images, the first image decodes, using Green, Red and Blue channels, rows first, to:

```
00000000  0d 03 00 00 00 00 1c 0c  0f 6d 74 76 5a 63 56 57  |.........mtvZcVW|
00000010  45 50 73 45 6f 5a 74 55  00 00 0c 0f 71 63 78 70  |EPsEoZtU....qcxp|
00000020  75 6a 5a 61 7a 71 50 45  43 4a 79 00 00 0c 0f 62  |ujZazqPECJy....b|
00000030  79 72 78 6f 69 6a 52 72  4e 63 6f 4e 67 75 00 00  |yrxoijRrNcoNgu..|
00000040  0c 12 ea b0 84 ea b0 94  ea b0 9f ea b0 8b ea b0  |................|
00000050  96 ea b0 a4 00 00 0c 0f  69 45 75 70 6b 7a 56 61  |........iEupkzVa|
00000060  77 63 6f 48 61 74 6f 00  00 0c 0f 76 44 77 46 77  |wcoHato....vDwFw|
00000070  55 6f 56 4c 58 71 4f 6f  67 61 07 0f 00 00 00 0c  |UoVLXqOoga......|
00000080  0f ea b0 82 ea b0 a3 ea  b0 89 ea b0 96 ea b0 81  |................|
00000090  00 00 0c 0f 78 66 6e 7a  79 4e 5a 6c 4a 74 46 69  |....xfnzyNZlJtFi|
000000a0  66 57 57 00 00 0c 0f 42  55 45 63 4e 4d 61 56 51  |fWW....BUEcNMaVQ|
000000b0  43 48 77 6b 50 6a 00 00  0c 0f 76 6a 6c 63 68 4e  |CHwkPj....vjlchN|
000000c0  7a 69 72 67 53 78 6a 53  43 07 00 00 00 00 0c 0f  |zirgSxjSC.......|
000000d0  45 70 6b 56 42 7a 74 4c  58 65 53 70 4b 77 65 02  |EpkVBztLXeSpKwe.|
000000e0  00 a0 01 00 4d 5a 90 00  03 00 00 00 04 00 00 00  |....MZ..........|
000000f0  ff ff 00 00 b8 00 00 00  00 00 00 00 40 00 00 00  |............@...|
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  f0 00 00 00 0e 1f ba 0e  00 b4 09 cd 21 b8 01 4c  |............!..L|
00000130  cd 21 54 68 69 73 20 70  72 6f 67 72 61 6d 20 63  |.!This program c|
00000140  61 6e 6e 6f 74 20 62 65  20 72 75 6e 20 69 6e 20  |annot be run in |
00000150  44 4f 53 20 6d 6f 64 65  2e 0d 0d 0a 24 00 00 00  |DOS mode....$...|
...
00000300  00 50 01 00 00 42 00 00  00 3c 01 00 00 00 00 00  |.P...B...<......|
00000310  00 00 00 00 00 00 00 00  40 00 00 40 2e 64 61 74  |........@..@.dat|
00000320  61 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |a...............|
00000330  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

This includes the start of a Windows executable.

The size of the chunk extracted from each image is stored in the first four bytes (DWORD), 0x30d in this case, less 15, and the required chunk of data starts at the 6th byte.

The chunks need to be assembled in numerical order of the resource names, which is different from the alphabetical order they appear in the file which is:

```
8286d0469740a3e495cfff46699c73c40
8286d0469740a3e495cfff46699c73c41
8286d0469740a3e495cfff46699c73c410
8286d0469740a3e495cfff46699c73c4100
8286d0469740a3e495cfff46699c73c4101
8286d0469740a3e495cfff46699c73c4102
8286d0469740a3e495cfff46699c73c4103
8286d0469740a3e495cfff46699c73c4104
8286d0469740a3e495cfff46699c73c4105
8286d0469740a3e495cfff46699c73c4106
8286d0469740a3e495cfff46699c73c4107
8286d0469740a3e495cfff46699c73c4108
8286d0469740a3e495cfff46699c73c4109
8286d0469740a3e495cfff46699c73c411
8286d0469740a3e495cfff46699c73c4110
8286d0469740a3e495cfff46699c73c4111
8286d0469740a3e495cfff46699c73c4112
8286d0469740a3e495cfff46699c73c4113
8286d0469740a3e495cfff46699c73c4114
8286d0469740a3e495cfff46699c73c4115
```

and the order they are referenced in the .NET metadata which is:

```
8286d0469740a3e495cfff46699c73c4120
8286d0469740a3e495cfff46699c73c4121
8286d0469740a3e495cfff46699c73c4122
8286d0469740a3e495cfff46699c73c4123
8286d0469740a3e495cfff46699c73c4124
8286d0469740a3e495cfff46699c73c4125
8286d0469740a3e495cfff46699c73c4126
8286d0469740a3e495cfff46699c73c4127
8286d0469740a3e495cfff46699c73c4128
8286d0469740a3e495cfff46699c73c4129
8286d0469740a3e495cfff46699c73c4102
8286d0469740a3e495cfff46699c73c4103
8286d0469740a3e495cfff46699c73c4100
8286d0469740a3e495cfff46699c73c4101
8286d0469740a3e495cfff46699c73c4106
8286d0469740a3e495cfff46699c73c4107
8286d0469740a3e495cfff46699c73c4104
8286d0469740a3e495cfff46699c73c4105
8286d0469740a3e495cfff46699c73c4108
8286d0469740a3e495cfff46699c73c4109
```

The reassembled payload in this case is Loki Bot Stealer.

In the following sample, SHA256 – 09c8cbd9cdfda1fcb7c6a051887213dc3e3ccf00a5877eca3d3e374f077b98d5, the images are BMPs and the first one looks like:

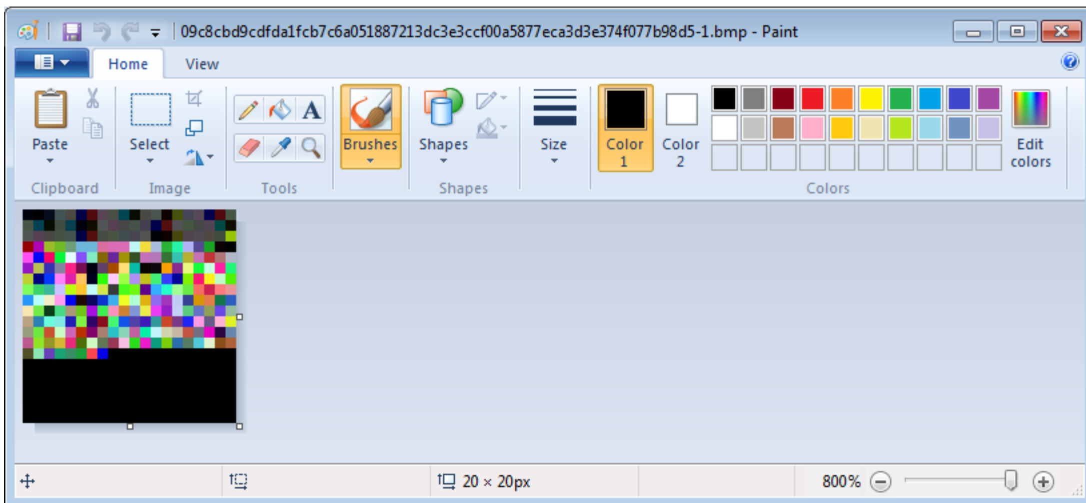

Figure 6: Image taken from sample SHA256: 09c8cbd9cdfda1fcb7c6a051887213dc3e3ccf00a5877eca3d3e374f077b98d5

The image decodes to the following, with chunk size highlighted in green, chunk data highlighted in yellow and blue:

```
00000000  0d 03 00 00 00 00 1c 0c  07 53 54 41 42 49 4c 49  |.........STABILI|
00000010  00 00 0c 09 50 52 45 56  45 4e 49 52 45 00 00 0c  |....PREVENIRE...|
00000020  06 43 48 49 4e 55 49 00  01 0c 09 52 45 56 45 44  |.CHINUI....REVED|
00000030  55 49 56 4d 00 00 0c 09  52 45 56 45 44 55 49 53  |UIVM....REVEDUIS|
00000040  42 00 00 0c 05 41 4d 41  4e 41 07 0f 00 00 00 0c  |B....AMANA......|
00000050  10 53 55 50 52 41 56 45  47 48 45 52 45 50 41 43  |.SUPRAVEGHEREPAC|
00000060  4b 00 00 0c 0f 53 55 50  52 41 56 45 47 48 45 52  |K....SUPRAVEGHER|
00000070  45 52 45 47 00 00 0c 10  53 55 50 52 41 56 45 47  |EREG....SUPRAVEG|
00000080  48 45 52 45 4e 53 45 49  00 00 0c 10 53 45 4c 45  |HERENSEI....SELE|
00000090  43 54 49 4f 4e 41 52 45  48 4f 53 54 07 03 00 00  |CTIONAREHOST....|
000000a0  00 0c 0e 46 49 4c 45 50  52 49 4e 43 49 50 41 4c  |...FILEPRINCIPAL|
000000b0  41 02 c5 97 02 00 94 b6  03 b0 28 bc 9a 25 ba 6d  |A.........(..%.m|
000000c0  7b 9f 6d db b6 6d db b6  6d 9d 6d db b6 6d db b6  |{.m..m..m.m..m..|
000000d0  6d db f3 f7 bd 3d dd f3  de bc aa 37 b3 2a a9 f5  |m....=.....7.*..|
000000e0  25 f9 b2 b2 92 4a 55 22  a3 19 07 00 0c 00 00 00  |%....JU"........|
000000f0  f2 4f fd fd 05 00 68 03  f8 37 f8 01 fe ff e1 ff  |.O....h..7......|
00000100  4f 85 c3 eb 80 03 68 82  9c 26 68 03 94 9e 26 50  |O.....h..&h...&P|
...
00000320  f9 eb 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000330  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

In this case, when assembled from the images, the payload is compressed using zlib Deflate, starting at byte 0xb0, highlighted in blue.

Decompressing gives:

```
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  6a 53 f9 5e 00 00 00 00  PE..L...jS.^....
```

which again is Agent Tesla in this case.

Conclusion

Generally, packers have different features that allow them to circumvent detection mechanisms by appearing as benign files, being difficult to reverse engineer, or incorporating sandbox evasion techniques. In this blog we've looked at two packers which use embedded images to hide the payload, one using a single image and the other using hundreds of them. These are just a few of the many tools threat actors have at their disposal to aid in distributing malware, collecting sensitive information, and gaining unauthorized access to systems.

IOCs

IOC	Type	Description
026b38e8eb0e4f505dc5601246143e7e77bbd2630b91df50622e7a14e0728675	SHA256	CyaX PNG sample with channels BGRA
c8c79ba04ab76c96db913f05b4b5bab36e7e0148fd72148df170a4be94d879a3	SHA256	Agent Tesla payload in 026b38e8eb0e4f505dc5601246143e7e77bbd2630t
083521fa4522245adc968b1b7dd18da29b193fd41572114c9d7dd927918234ea	SHA256	CyaX PNG sample with gzipped data
a6f7edd2654412c25d7c565cb5b52e1382799a8b86d6bc44e965b554f6344618	SHA256	Agent Tesla payload in 083521fa4522245adc968b1b7dd18da29b193fd415
04794ec7e7eb5c6611aada660fb1716a91e01503fb4703c7d2f2099c089c9017	SHA256	CyaX PNG sample with double steganography

6d9c861bf6f1495a4bddc7c745eb5b504692b4d6eae31e89453f0829760b1b90	SHA256	Remcos RAT payload in 04794ec7e7eb5c6611aada660fb1716a91e01503fb
0091c6bdceecf3e0143b4eaaefca1cd56cbfdfc55f99c167f9dd1f3a48928bb5	SHA256	Hectobmp sample with PNGs
1180c158968faaf0a4951e9a0c59996f0fb29cdad9443aa2097efb5bc7f123f4	SHA256	Loki Bot payload in 0091c6bdceecf3e0143b4eaaefca1cd56cbfdfc55f99
09c8cbd9cdfda1fcb7c6a051887213dc3e3ccf00a5877eca3d3e374f077b98d5	SHA256	Hectobmp sample with BMPs
c3b85d8291281d73cfdd8373cb2b32cdc4c3a602233f99ab3cbbd34bd4e3c99b	SHA256	Agent Tesla payload in 09c8cbd9cdfda1fcb7c6a051887213dc3e3ccf00a58

References

De4dot
ILSpy
Agent Tesla: A day in a life of IR, Full description of an Agent Tesla campaign using CyaX packer (steganographic variant)

Subscribe to the Proofpoint Blog