{"®eve®se": "Enginee®ing"}

Published: 2021-01-18 · Archived: 2026-04-05 16:29:06 UTC

In this past few days I stumble to some new and old variant of ICEID malware that uses .png steganography to hide and execute its encrypted shellcode. In this article I will share how the structure of the Iceid png payload look like and how to extract its encrypted shellcode.

Loader Compression:

same as the other malware, IceID Loader changes its Crypter to execute its main module in memory. From old variant where the encrypted code stub and decryption module is place in the RSRC section as a data .rsrc entry (RC4 encrypted) to applib compression like the figure below.

```
00000000: 4d38 5a90 3803 6602 0409 71ff 81b8 c291  M8Z.8.f...q.....
00000010: 0140 c215 c6d0 091c 0e1f baf8 00b4 09cd  .@..............
00000020: 21b8 014c c00a 5468 6973 200e 7072 6f67  !..L..This .prog
00000030: 6761 6d87 6347 6e1f 4f74 e762 65af cf75  gam.cGn.Ot.be..u
00000040: 5f98 6906 444f 7e53 036d 6f64 652e 0d89  _.i.DO~S.mode...
00000050: 0a24 4c44 9d01 fb7b 6ed9 9a15 3d58 04aa  .$LD...{n...=X..
00000060: 0af8 143c d40c bc7c 60ff 113f fe1d 3c43  ...<...|`..?..<C
00000070: de7c f0d8 8643 1705 5269 6368 1030 04a8  .|...C..Rich.0..
00000080: 5040 454c 3801 0505 9897 c05f 6314 e080  P@EL8......_c...
00000090: 0221 0b1c 010e 0c1b 101b b609 ff01 d41a  .!..............
000000a0: 044a [illegible] 08a7  .J. #....Lt..#..
000000b0: 0460 9d1d 121f 4093 3542 5808 d007 8670  .`....@.5BX....p
000000c0: 31f9 c811 c092 787a 3567 4bd0 0a01 d599  1.....xz5gK.....
000000d0: 3006 9c5a 1fc1 2e74 6578 ce22 9f0f bb91  0..Z...tex."....
000000e0: a539 b819 0110 eb60 6273 e3ab 1807 b422  .9.....`bs....."
000000f0: 3a4c 2a80 c707 2e72 6461 7428 0a5a 057c  :L*....rdat(.Z.|
00000100: 4e08 0691 14e3 4029 072e 2720 d74d 0235  N.....@)..' .M.5
00000110: d074 2b1a 28ec a165 156c 6f63 fcc9 5033  .t+.(..e.loc..P3
00000120: 6c82 1ebe 2870 42a7 01df 6083 ec14 5300  l...(pB...`...S.
00000130: 5556 578b f96a 085e ea2f 1f85 ed74 bd9d  UVW..j.^./...t..
00000140: 5f04 dddb c04c 3bde 7248 833a fb10 8743  _....L;.rH.:...C
00000150: 8b0c 2e31 d181 e275 1fc3 dbc1 88e8 100b  ...1...u........
00000160: 67d0 0fe0 7381 e170 23fb 8d16 ea08 1ccf  g...s..p#.......
00000170: 002a 8d42 0c03 c63b c300 7713 817c 2e04  .*.B...;..w..|..
-- More  --
```



```
00000010: b800 0000 0000 0000 4000 0000 0000 0000  ........@.......
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 d000 0000  ................
00000040: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468  ........!..L.!Th
00000050: 6973 2070 726f 6772 616d 2063 616e 6e6f  is program canno
00000060: 7420 6265 2072 756e 2069 6e20 444f 5320  t be run in DOS 
00000070: 6d6f 6465 2e0d 0d0a 2400 0000 0000 0000  mode....$.......
00000080: 9dfb 7b6e d99a 153d d99a 153d d99a 153d  ..{n...=...=...=
00000090: aaf8 143c d49a 153d d99a 143d ff9a 153d  ...<...=...=...=
000000a0: 3ffe [illegible] 153d  ?..<...=?..<...=
000000b0: 3ffe 173c d89a 153d 5269 6368 d99a 153d  ?..<...=Rich...=
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 5045 0000 4c01 0500 9897 c05f 0000 0000  PE..L......_....
000000e0: 0000 0000 e000 0221 0b01 0e0c 0010 0000  .......!........
000000f0: 000c 0000 0002 0000 d41a 0000 0010 0000  ................
00000100: 0020 0000 0000 0010 0010 0000 0002 0000  . ..............
00000110: 0500 0100 0000 0000 0500 0100 0000 0000  ................
00000120: 0060 0000 0004 0000 0000 0000 0200 4000  .`............@.
00000130: 0000 1000 0010 0000 0000 1000 0010 0000  ................
00000140: 0000 0000 1000 0000 7031 0000 5000 0000  ........p1..P...
00000150: c031 0000 7800 0000 0000 0000 0000 0000  .1..x...........
00000160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000170: 0050 0000 d000 0000 0000 0000 0000 0000  .P..............
-- More  --
```


figure 1: The aplib decompressed module of ICEID

PNG Header:

Before we deal with the ICEID PNG steganography, I think it is a good idea to have some preview what PNG file header format is. It will give as a clear preview how ICEID parse the .PNG header and look for its encrypted shellcode.

The PNG file format started with **8-bytes signature header "89 50 4e 47 0d 0a 1a 0a"**. after this header is a chunk structure or series of chunk structures that contains either a **"critical chunks"** like "IHDR", "IDAT" and etc... or **"Ancillary chunks"** that may contain some attribute related to the color, pixel or metadata of the png file like "sRGB", "gAMA" and many more.

each PNG chunks layout consist of 4 parts or 4 structure member. the figure below show the 4 parts in IHDR chunk type.

figure 2: PNG File header Format

ICEID PNG Module Decryptor:

Now that we have some insight how PNG file format look like, lets dive in to the ICEID PNG decryptor module **(*"let's call it PNG module"*)** that we already extracted earlier in the memory. This part is really interesting especially in parsing the header. :)

The PNG module start by executing a thread that will do the following:

1. decrypt its data section (C&C url link) using same approach how it decrypt the shellcode in PNG payload file.

2. it will check the existence of the PNG file in %appdata%<randomname>/, if not

3. it will try to download it to its C&C server

4. then it will parse and check the png file to extract and execute its shellcode.

The first task is decrypting the C&C server URL link that are place in data section that are encrypted in RC4 algorithm. the structure of data it used in decrypting this data section can be seen before a call function that will do the decryption part.

```
push    ebp
mov     ebp, esp
sub     esp, 128h
mov     eax, offset dword_10004008
mov     [ebp+var_24], offset unk_10004000
lea     ecx, [ebp+var_24]
mov     [ebp+var_20], 8
mov     [ebp+var_1C], eax
mov     [ebp+var_18], 248h
mov     [ebp+var_14], eax
call    func_DecryptDataBlock
```

```
00000000 ; U       : delete structure member
00000000 ; ---------------------------------------
00000000
00000000 EncryptedBlobStruct struc ; (sizeof=0x14,
00000000 DecryptionKeyVirtualAddrs dd ?       ;
00000004 DecryptionKeySize dd ?               ;
00000008 EncryptedBLobVirtualAddrs2 dd ?      ;
0000000C EncyptedBlobSize dd ?                ;
00000010 EncryptedBLobVirtualAddrs dd ?       ;
00000014 EncryptedBlobStruct ends
00000014
00000000 ; ---------------------------------------
```

figure 3: Encrypted Data Structure

The first 8 bytes of the encrypted data section is the RC4 key and the rest is the encrypted data.

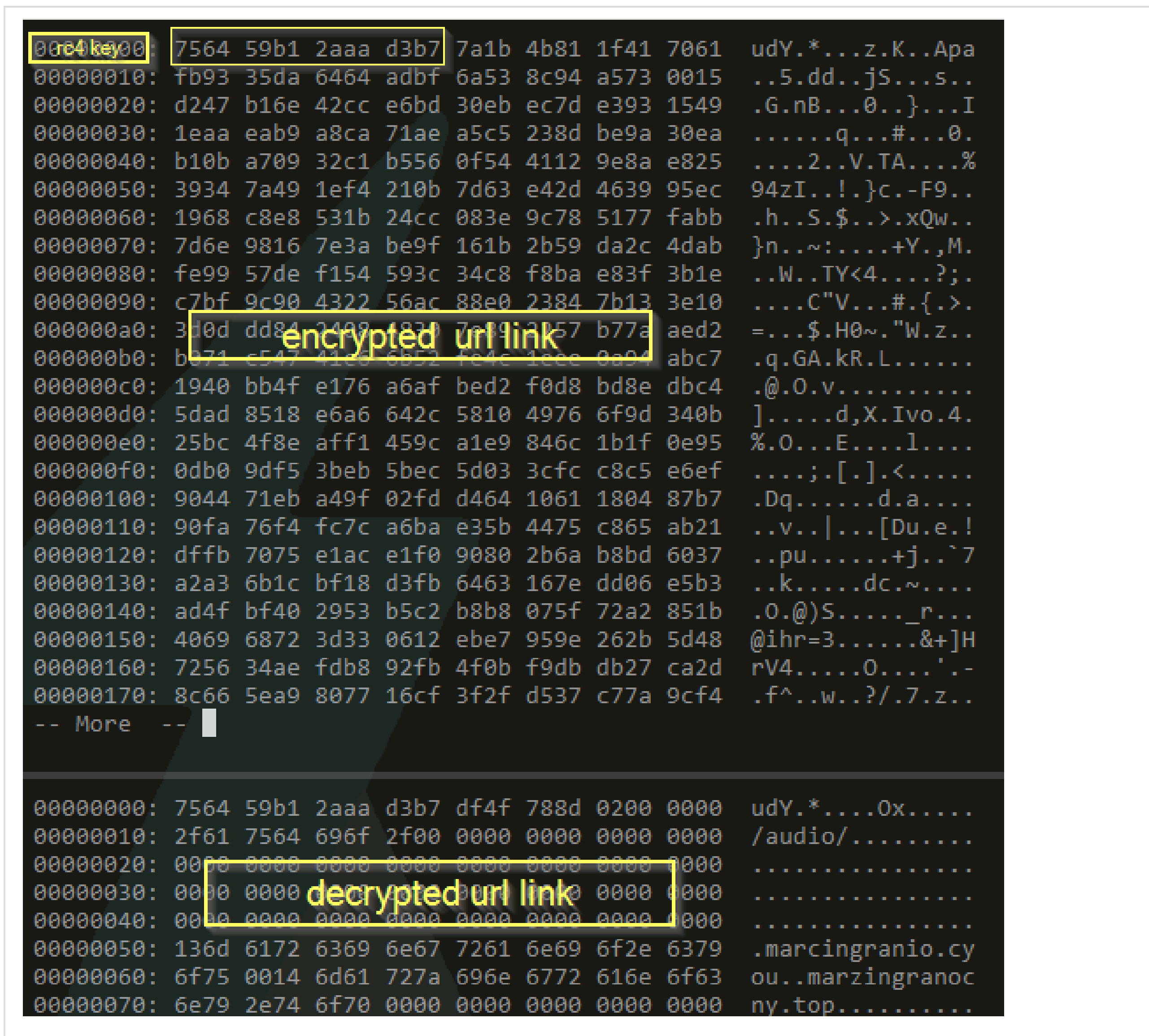

figure 4: decrypting C&C server URL link

Next it will create a random name folder in %appdata% using the "username" of the infected machine.

with the use of RDTSC command to generate random character. If the module didn't find the png payload in the said folder it will try to contact the C&C server to download it.

```
lpBufferPngFile = (LPCVOID)dword_10004008;
appdata_path = func_CreateAppDataDir(FileName);
func_StrConcat(1u, (int *)&lpBufferPngFile, ".png", &FileName[appdata_path]);
if ( !func_ReadPngPayload(FileName, (void **)&lpBufferPngFile, (int)&nNumberOfBytesToWrite)
  || !func_LocatePayloadHeader(nNumberOfBytesToWrite, (int)lpBufferPngFile, (LPVOID *)&v6, v5) )
{
  if ( !func_DownloadPngPayload((void **)&lpBufferPngFile, &nNumberOfBytesToWrite)
    || !func_LocatePayloadHeader(nNumberOfBytesToWrite, (int)lpBufferPngFile, (LPVOID *)&v6, v5) )
  {
    return 0;
```

figure 5: looking for the PNG payload file

Parsing PNG Header:

after reading the PNG and save it in the memory, it will start the checking in offset 0x8 (skipping the PNG header) which is the *"chunk_data_length"* of the first chunk type in the header which in our case is "IHDR". The way how it parse the header to look for "IDAT" chunk_type structure is by adding:

next_chunk_type_struct = size(chunk_data) + chunk_data_length (4 bytes) + chunk_type (4byes) + chunk_crc32 (4 bytes)

except for the start **chunk_type** structure where you need to include or add the PNG header size which 8 bytes.

for this topic, I created a simple python script that will parse this header and give you the basic information about the header. it also parse the **chunk_data** but I place it in the **debug.log** of this script.

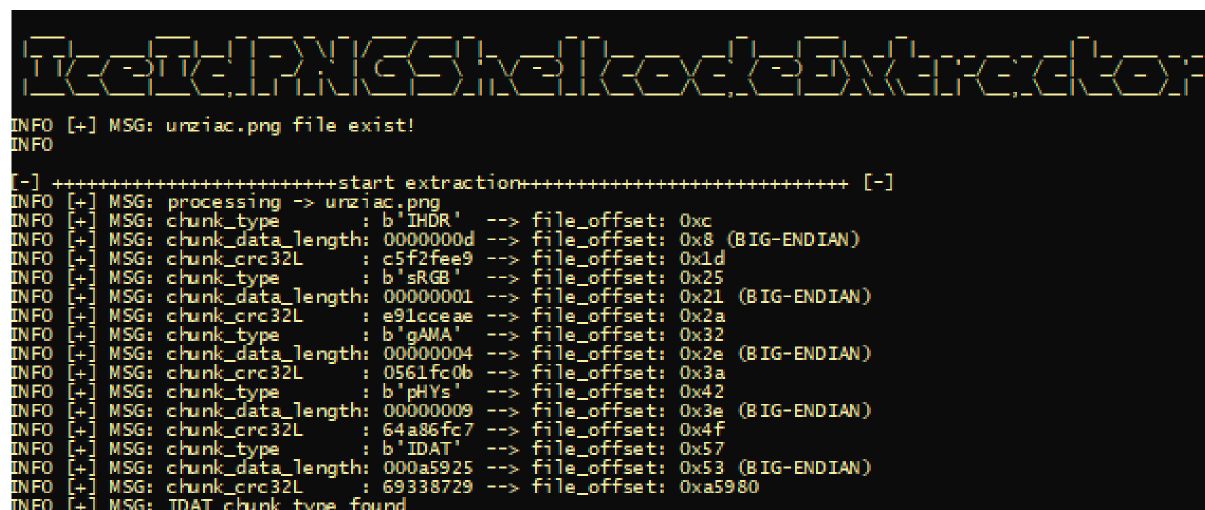

figure 6: parsing the header

Byte Flag and Decryption Key:

as soon as it found the **IDAT chunk_type** structure it will check first if the **chunk_data_length > 5** then it will skip the **chunk_data_length** , **chunk_type** and **chunk_crc32** by adding **0x0c** to the current pointer of the "IDAT" chunk header. the byte in this position looks like a validity flag of the PNG. If this byte is zero, the PNG module will check another value which is one of the parameter to the function that parse the png header which is also zero, so in this case it will exit the flag.

```
; ---------------------------------------------------------------------------

loc_10001070:                           ; CODE XREF: func_ParsingPNGHdr+5A↑j
                lea     ecx, [esi+ebp]
                cmp     edx, 5
                jb      short exitfunc
                mov     ebp, [ecx+8]
                sub     edx, 4
                add     ecx, 0Ch
                mov     al, [ecx]
                movzx   esi, al
                cmp     esi, edx
                ja      short exitfunc
                test    al, al          ; byte flag check
                jnz     short loc_100010A7
                mov     eax, [edi+8]    ; parameter which is also zero
                test    eax, eax
                jz      short exitfunc
                mov     esi, [edi+0Ch]
                test    esi, esi
                jz      short exitfunc
```

figure 7: byte flag

after this byte is the 8 byte RC4 decryption key followed by the encrypted shellcode using this RC4 key. the python script I mentioned earlier will parse the RC4 key, extract the shellcode and check the shellcode header and entrypoint by dis-assembling it using capstone python library.

```
INFO [+] MSG: IDAT chunk_type found
INFO [+] MSG: checking the iceid idat data valid byte flag....
INFO [+] MSG: non-zero idat chunk data byte flag ...
INFO [+] MSG: RC4 decryption key: b'af17555297c52b88' dec_key_size: 0x8
INFO [+] MSG: enc_data_size: 0xa5918
INFO [+] MSG: enc_data_ofs: 0x68
INFO [+] MSG: RC4 0x100 byte swap table
---------------------------------------
INFO []
INFO ['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '0a', '0b', '0c', '0d', '0e', '0f']
INFO ['10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '1a', '1b', '1c', '1d', '1e', '1f']
INFO ['20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '2a', '2b', '2c', '2d', '2e', '2f']
INFO ['30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '3a', '3b', '3c', '3d', '3e', '3f']
INFO ['40', '41', '42', '43', '44', '45', '46', '47', '48', '49', '4a', '4b', '4c', '4d', '4e', '4f']
INFO ['50', '51', '52', '53', '54', '55', '56', '57', '58', '59', '5a', '5b', '5c', '5d', '5e', '5f']
INFO ['60', '61', '62', '63', '64', '65', '66', '67', '68', '69', '6a', '6b', '6c', '6d', '6e', '6f']
INFO ['70', '71', '72', '73', '74', '75', '76', '77', '78', '79', '7a', '7b', '7c', '7d', '7e', '7f']
INFO ['80', '81', '82', '83', '84', '85', '86', '87', '88', '89', '8a', '8b', '8c', '8d', '8e', '8f']
INFO ['90', '91', '92', '93', '94', '95', '96', '97', '98', '99', '9a', '9b', '9c', '9d', '9e', '9f']
INFO ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 'a8', 'a9', 'aa', 'ab', 'ac', 'ad', 'ae', 'af']
INFO ['b0', 'b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8', 'b9', 'ba', 'bb', 'bc', 'bd', 'be', 'bf']
INFO ['c0', 'c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8', 'c9', 'ca', 'cb', 'cc', 'cd', 'ce', 'cf']
INFO ['d0', 'd1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8', 'd9', 'da', 'db', 'dc', 'dd', 'de', 'df']
INFO ['e0', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6', 'e7', 'e8', 'e9', 'ea', 'eb', 'ec', 'ed', 'ee', 'ef']
INFO [+] MSG: RC4 KSA table using the dec key
---------------------------------------
INFO []
INFO ['f2', '32', '57', '91', '0e', '75', '09', '72', '4f', '6c', '64', '28', '51', '96', '6a', 'ac']
INFO ['c5', 'b9', '42', '4d', 'c1', '9b', '97', '01', '3f', '6f', 'de', 'c9', '8f', 'e0', '86', 'b4']
INFO ['94', '5e', '83', '79', '50', 'a2', '8b', '1b', 'f4', '34', '14', 'c3', 'f3', 'e5', '4c', '11']
INFO ['cf', '78', '80', '7c', '39', 'dd', '2e', 'f9', '6d', 'e3', 'b0', '3d', '22', '1f', '89', '4a']
INFO ['36', '27', 'd5', 'd6', '5f', '69', 'da', 'a9', 'ee', '40', '66', 'd3', 'b6', 'c8', '58', '45']
INFO ['a4', '0a', '2a', 'ce', '61', '5d', '05', 'bd', '2d', '0d', 'cd', '90', '37', 'f7', 'bb', 'ed']
INFO ['5c', '74', '30', '8c', '87', '8d', '38', 'dc', '06', 'd1', 'cc', '5a', '2c', 'fb', '95', 'd4']
INFO ['68', 'af', 'a8', '7e', '84', '7d', '99', '3e', '0b', '43', '41', '47', '6e', '8e', '60', '73']
INFO ['13', 'ad', 'a1', '16', '33', 'aa', '6b', '52', '4e', 'e1', 'c0', '9c', '20', '12', '53', 'fe']
INFO ['26', '7f', 'ea', '00', 'fa', '48', 'a3', 'b2', 'c6', '59', '08', '9f', 'd2', '24', '1d', 'c7']
INFO ['e2', '77', 'd7', '1c', '65', '92', 'ec', 'a5', '19', 'd0', '93', '04', '63', 'ca', '56', '23']
INFO ['b8', 'f0', '29', 'cb', '3c', '8a', '82', '0c', '35', 'f1', '7b', '2b', 'e6', '2f', 'e9', '55']
INFO ['ff', 'b3', 'd9', '62', '9d', '3b', 'd8', '31', 'ab', '17', '70', 'c4', '0f', 'ae', '7a', '9a']
INFO ['10', 'e4', 'df', 'f8', '98', 'ef', '54', '3a', '1a', '9e', 'a7', 'a0', 'a6', 'bc', 'f6', '85']
INFO ['15', '71', '21', 'c2', '44', 'bf', '88', '25', 'eb', 'e8', '4b', 'b7', 'f5', '76', 'db', 'b1']
INFO [+] MSG: DWORD (little) shellcode_header: 01020100
INFO [+] MSG: DWORD (little) total_encrypted_size: 000a5918
INFO [+] MSG: DWORD (little) shellcode_entrypoint: 0000009c
INFO [+] MSG: DWORD (little) shellcode_size: 0000151f
INFO [+] MSG: shellcode extracted: urziac.png_shellcode.bin
---------------------------------------

INFO [+] MSG: Disassembling first 0x100 bytes of shellcode starting shellcode oep
---------------------------------------

INFO 0x0004009c:        push    ebp
INFO 0x0004009d:        mov     ebp, esp
INFO 0x0004009f:        sub     esp, 0x470
INFO 0x000400a5:        cmp     dword ptr [ebp + 8], 0
INFO 0x000400a9:        push    ebx
INFO 0x000400aa:        push    esi
INFO 0x000400ab:        push    edi
INFO 0x000400ac:        je      0x40324
INFO 0x000400b2:        mov     eax, dword ptr fs:[0x30]
INFO 0x000400b8:        xor     ebx, ebx
INFO 0x000400ba:        test    eax, eax
INFO 0x000400bc:        je      0x40324
INFO 0x000400c2:        mov     eax, dword ptr [eax + 0xc]
INFO 0x000400c5:        mov     ecx, dword ptr [eax + 0x1c]
INFO 0x000400c8:        test    ecx, ecx
INFO 0x000400ca:        je      0x40324
INFO 0x000400d0:        add     ecx, -0x10
INFO 0x000400d3:        je      0x40324
INFO 0x000400d9:        mov     eax, dword ptr [ecx + 0x28]
INFO 0x000400dc:        test    eax, eax
INFO 0x000400de:        je      0x400e7
INFO 0x000400e0:        mov     al, byte ptr [eax]
INFO 0x000400e2:        mov     byte ptr [ebp - 5], al
INFO 0x000400e5:        jmp     0x400eb
INFO 0x000400e7:        mov     byte ptr [ebp - 5], 0x43
INFO 0x000400eb:        mov     ecx, dword ptr [ecx + 0x18]
INFO 0x000400ee:        test    ecx, ecx
INFO 0x000400f0:        je      0x40324
INFO 0x000400f6:        mov     eax, 0x5a4d
INFO 0x000400fb:        cmp     word ptr [ecx], ax
INFO 0x000400fe:        jne     0x40324
INFO 0x00040104:        mov     esi, dword ptr [ecx + 0x3c]
```

figure 8: script tool parser

Conclusion:

In this analysis we learned how PNG file can be used as a weapon to hide the malicious code and how malware keeps on updating their tools to bypassed detection.

Also thanks to the community for sharing the samples :)

Samples:

sha1: 9a07f8513844e7d3f96c99024fffe6e891338424

sha1: 1ab6006621c354649217a011cd7ca8eb357c3db4

sha1: c1faa9cb4aa7779028008375e7932051ee786a52

sha1: 481bc0cbdcae1cd40b70b93388bf4086781b44b4

https://www.virustotal.com/gui/file/45520a22cdf580f091ae46c45be318c3bb4d3e41d161ba8326a2e29f30c025d4/details

https://www.virustotal.com/gui/file/e6e0adcc94c3c4979ea1659c7125a11aa7cdabe24a36f63bfe1f2aeee2c5d3a1/detection

https://www.virustotal.com/gui/file/cc1030c4c7486f5295444acb205fa9c9947ad41427b6b181d74e7e5fe4e6f8a9/details

https://www.virustotal.com/gui/file/f6ea81aaf9a07e24a82b07254a8ed4fcf63d5a8e6ea7b57062f4c5baf9ef8bf2/detection

References:

https://en.wikipedia.org/wiki/Portable_Network_Graphics

http://www.libpng.org/pub/png/spec/1.2/PNG-Structure.html

https://blog.malwarebytes.com/threat-analysis/2019/12/new-version-of-icedid-trojan-uses-steganographic-payloads/

https://www.malware-traffic-analysis.net/2020/12/11/index.html

---

Source: https://tccontre.blogspot.com/2021/01/