# Extracting Shellcode in ICEID .PNG Steganography

tccontre.blogspot.com/2021/01/

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.

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.

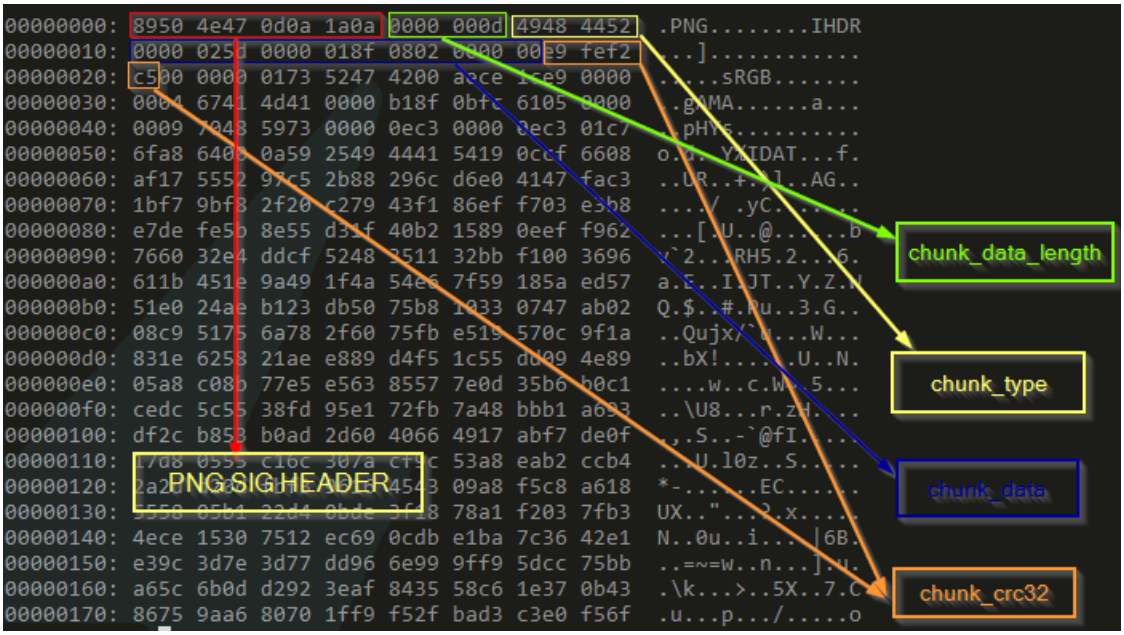


figure 2: PNG File header Format

## 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.


```
00000000: 7564 59b1 2aaa d3b7 7a1b 4b81 1f41 7061  udY.*...z.K..Apa
00000010: fb93 35da 6464 adbf 6a53 8c94 a573 0015  ..5.dd..jS...s..
00000020: d247 b16e 42cc e6bd 30eb ec7d e393 1549  .G.nB...0..}...I
00000030: 1eaa eab9 a8ca 71ae a5c5 238d be9a 30ea  ......q...#...0.
00000040: b10b a709 32c1 b556 0f54 4112 9e8a e825  ....2..V.TA....%
00000050: 3934 7a49 1ef4 210b 7d63 e42d 4639 95ec  94zI..!.}c.-F9..
00000060: 1968 c8e8 531b 24cc 083e 9c78 5177 fabb  .h..S.$..>.xQw..
00000070: 7d6e 9816 7e3a be9f 161b 2b59 da2c 4dab  }n..~:....+Y.,M.
00000080: fe99 57de f154 593c 34c8 f8ba e83f 3b1e  ..W..TY<4....?;.
00000090: c7bf 9c90 4322 56ac 88e0 2384 7b13 3e10  ....C"V...#.{.>.
000000a0: 3d0d dd84 [illegible] 7d57 b77a aed2  =...$.H0~."W.z..
000000b0: [illegible] abc7  .q.GA.kR.L......
000000c0: 1940 bb4f e176 a6af bed2 f0d8 bd8e dbc4  .@.O.v..........
000000d0: 5dad 8518 e6a6 642c 5810 4976 6f9d 340b  ].....d,X.Ivo.4.
000000e0: 25bc 4f8e aff1 459c a1e9 846c 1b1f 0e95  %.O...E....l....
000000f0: 0db0 9df5 3beb 5bec 5d03 3cfc c8c5 e6ef  ....;.[.].<.....
00000100: 9044 71eb a49f 02fd d464 1061 1804 87b7  .Dq......d.a....
00000110: 90fa 76f4 fc7c a6ba e35b 4475 c865 ab21  ..v..|...[Du.e.!
00000120: dffb 7075 e1ac e1f0 9080 2b6a b8bd 6037  ..pu......+j..`7
00000130: a2a3 6b1c bf18 d3fb 6463 167e dd06 e5b3  ..k.....dc.~....
00000140: ad4f bf40 2953 b5c2 b8b8 075f 72a2 851b  .O.@)S....._r...
00000150: 4069 6872 3d33 0612 ebe7 959e 262b 5d48  @ihr=3......&+]H
00000160: 7256 34ae fdb8 92fb 4f0b f9db db27 ca2d  rV4.....O....'.-
00000170: 8c66 5ea9 8077 16cf 3f2f d537 c77a 9cf4  .f^..w..?/.7.z..
-- More  --
```

```
00000000: 7564 59b1 2aaa d3b7 df4f 788d 0200 0000  udY.*....Ox.....
00000010: 2f61 7564 696f 2f00 0000 0000 0000 0000  /audio/.........
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 136d 6172 6369 6e67 7261 6e69 6f2e 6379  .marcingranio.cy
00000060: 6f75 0014 6d61 727a 696e 6772 616e 6f63  ou..marzingranoc
00000070: 6e79 2e74 6f70 0000 0000 0000 0000 0000  ny.top.........
```

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.

https://github.com/tccontre/KnowledgeBase/tree/main/malware_re_tools/iceid_stego_shell_decryptor

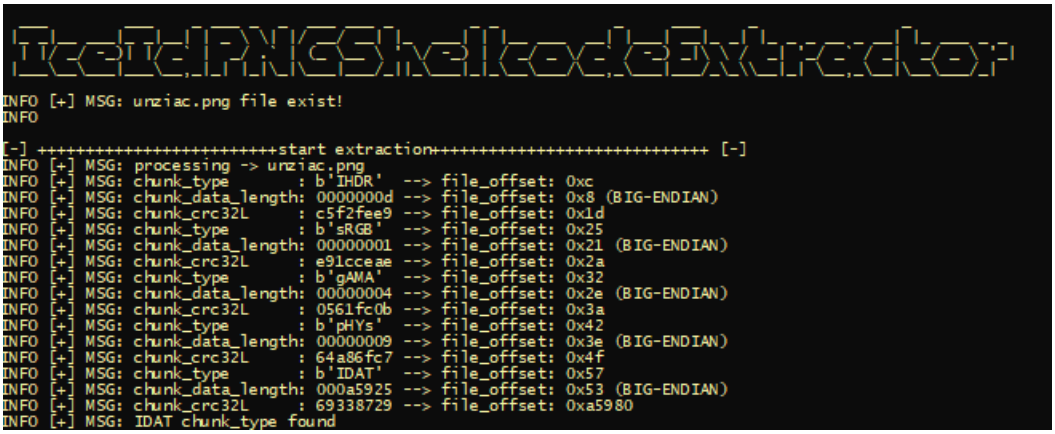


figure 6: parsing the header

## 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.

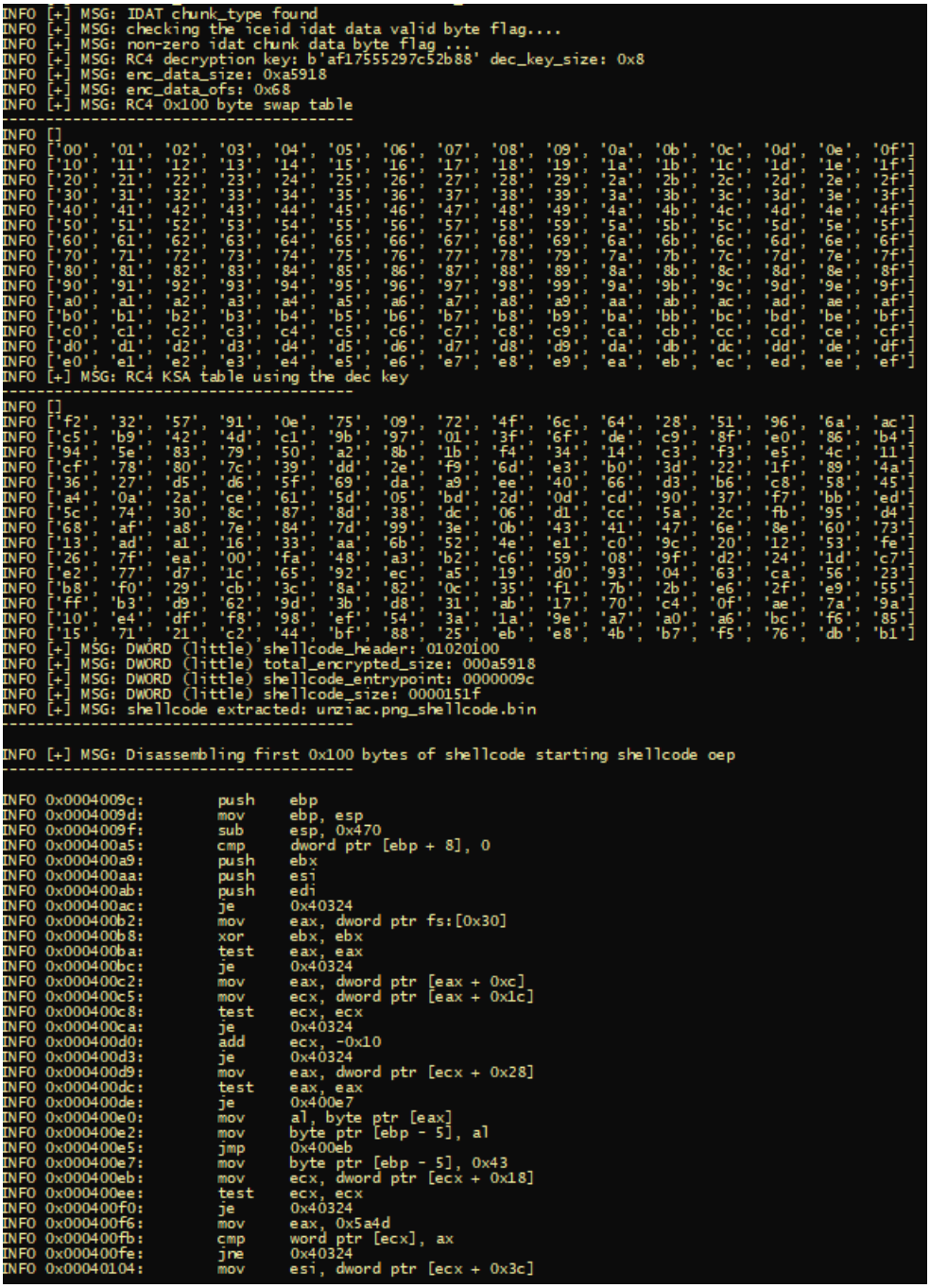


figure 8: script tool parser

## 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