Turla PNG Dropper is back

22 November 2018 By Matt Lewis

Research Research Reverse Engineering Vulnerability Threat Intelligence

This is a short blog post on the PNG Dropper malware that has been developed and used by the Turla Group [1]. The PNG Dropper was first discovered back in August 2017 by Carbon Black researchers. Back in 2017 it was being used to distribute Snake, but recently NCC Group researchers have uncovered samples with a new payload that we have internally named RegRunnerSvc.

It's worth noting at this ged to obtain. There will be a first stage drop e think that this it is worth documenting thi

PNG Dropp

The PNG Dropper comp k [1], but for the purpose of clarity we w

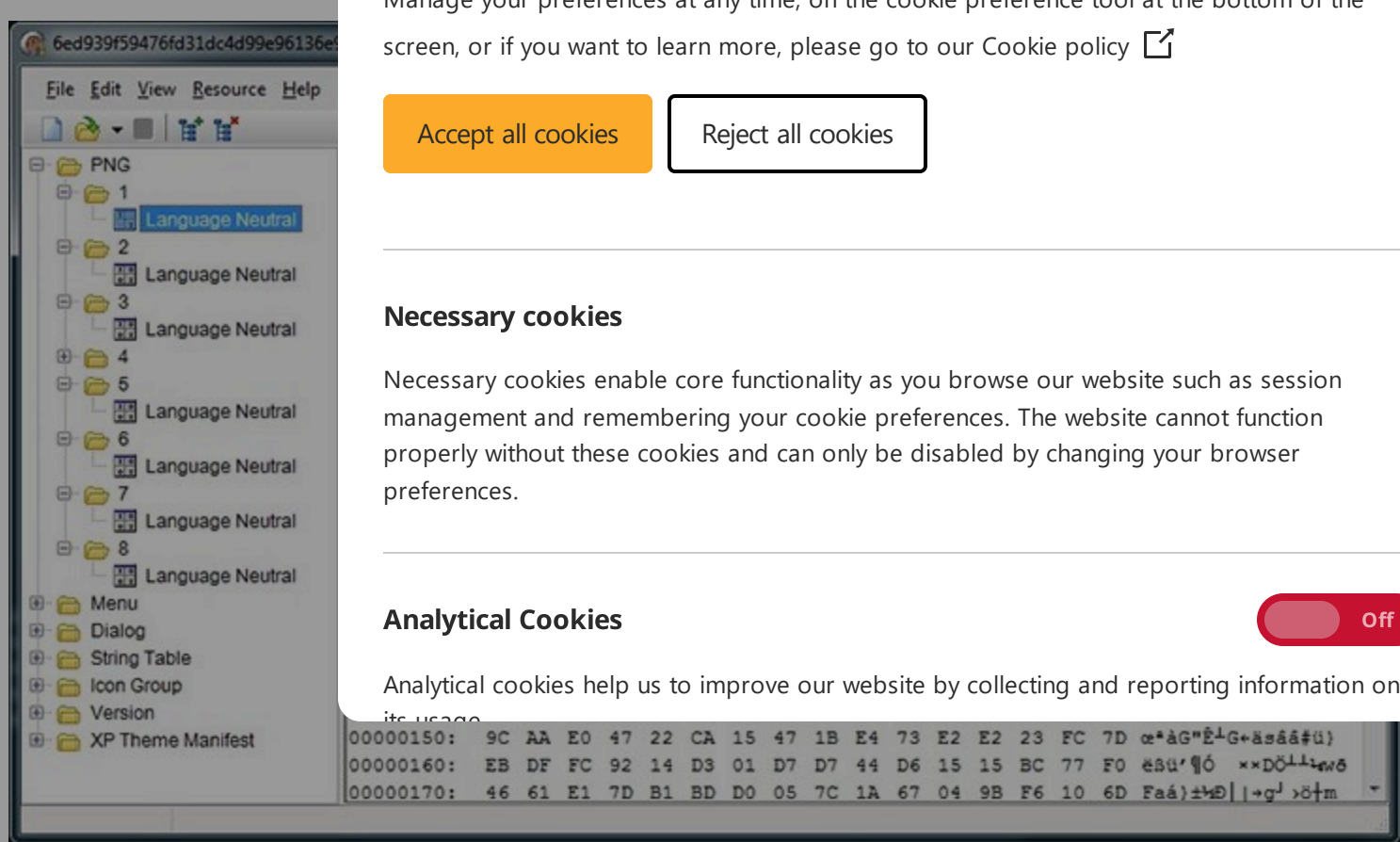

Figure 1

The purpose of the dropper is to load and run a PE file that is hidden in a number of PNG files. Figure 1 shows the resources of the dropper. Here you can see a number binary data resource entries under the name "PNG". Each of these resources is a valid PNG file which can be viewed with any image viewer, but upon opening one you will only see a few coloured pixels (see an enlarged version in Figure 2).

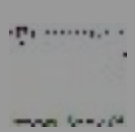

Figure 2

The PNG is loaded using Microsoft's GDI+ library. In Figure 3 we see a call to LockBits which is used to read the pixel data from the PNG file. Each byte in the pixel data represents an RGB value for a pixel. Encoded in each of the the RGB values is a byte from a PE file. It doesn't make for a very meaningful image, but it is a novel way to hide data in seemingly innocent resources.

```
.text:000000013F700D26 8B D0                    mov     edx, eax        ; dwSize
.text:000000013F700D28 49 89 17                 mov     [r15], rdx
.text:000000013F700D2B 33 C9                    xor     ecx, ecx        ; lpAddress
.text:000000013F700D2D 44 8D 49 40              lea     r9d, [rcx+40h]  ; flProtect
.text:000000013F700D31 41 B8 00 10 00 00        mov     r8d, 1000h      ; flAllocationType
.text:000000013F700D37 FF 15 CB 06 FF FF        call    cs:VirtualAlloc
.text:000000013F700D3D 4C 8B 6D 6F              mov     r13, [rbp+57h+arg_8]
.text:000000013F700D41 49 89 45 00              mov     [r13+0], rax
.text:000000013F700D45 48 85 C0                 test    rax, rax
.text:000000013F700D48 0F 84 EC 00 00 00        jz      loc_13F700E3A
.text:000000013F700D4E 48 89 45 FF              mov     [rbp+57h+var_58], rax
.text:000000013F700D52 48 8D 45 EF              lea     rax, [rbp+57h+var_68]
.text:000000013F700D56 48 89 44 24 20           mov     [rsp+0C0h+var_A0], rax
.text:000000013F700D5B 41 B9 07 10 06 00        mov     r9d, 61007h
.text:000000013F700D61 41 B8 07 00 00 00        mov     r8d, 7
.text:000000013F700D67 48 8D 55 D7              lea     rdx, [rbp+57h+var_80]
.text:000000013F700D6B 48 8B 4F 08              mov     rcx, [rdi+8]
.text:000000013F700D6F E8 D0 9C 01 00           call    GdipBitmapLockBits
.text:000000013F700D74 8B F0                    mov     esi, eax
.text:000000013F700D76 85 C0                    test    eax, eax
.text:000000013F700D78 74 05                    jz      short loc_13F700D7F
.text:000000013F700D7A 89 47 10                 mov     [rdi+10h], eax
.text:000000013F700D7D EB 02                    jmp     short loc_13F700D81
```

Figure 3

Each PNG resource is enumerated and the pixel data is extracted and then concatenated together. The result is an entire PE file contained in memory. The dropper will then manually load the PE file. The imports are processed, as are the relocations. Finally the PE file's entry point is executed (as shown in Figure 4).

```
.text:000000013F700FE4 48 63 43 3C              movsxd  rax, dword ptr [rbx+3Ch]
.text:000000013F700FE8 B9 08 01 00 00           mov     ecx, 108h
```


RegRunnerS

The PNG dropper will d… erSvc is to extract an encrypted payload fron… we have not managed to obtain) will have alre…

Figure 5 shows the entr… r. In this case the name of the service is WerFau… ows Error Reporting service. The service also…

```
push rbx
sub rsp,40
mov rbx,rcx
mov ecx,8007
call qword ptr ds:[<&SetE
cmp byte ptr ds:[1400040
mov qword ptr ds:[140004
lea r8,qword ptr ds:[140
je 6e.140001BA2
mov rcx,r8
mov edx,20
movzx eax,byte ptr ds:[r
add rcx,4
xor byte ptr ds:[rcx-4],
movzx eax,byte ptr ds:[r
xor byte ptr ds:[rcx-3],
movzx eax,byte ptr ds:[r
xor byte ptr ds:[rcx-2],
movzx eax,byte ptr ds:[rcx-81]
xor byte ptr ds:[rcx-1],al
sub rdx,1
jne 6e.140001B73
lea rax,qword ptr ds:[140001A80]
lea rcx,qword ptr ss:[rsp+20]                    [rsp+20]:"WerFaultSvc"
mov qword ptr ss:[rsp+20],r8                     [rsp+20]:"WerFaultSvc"
mov qword ptr ss:[rsp+28],rax
xor eax,eax
mov qword ptr ss:[rsp+30],rax
mov qword ptr ss:[rsp+38],rax
call qword ptr ds:[<&StartServiceCtrlDispatcherA>]
xor eax,eax
add rsp,40
pop rbx
ret
```

Figure 5

After the service setup functions has been executed, it is time to find the data in the registry. Generally the path to the registry value would be stored as a (possibly encrypted/obfuscated) string within the binary, but interestingly this is not the case here. The registry keys and values are enumerated using the RegEnumKeyExA and RegEnumValueA functions. The enumeration starts at the root of the HKEY_LOCAL_MACHINE key and continues using a depth first search until either the data is found or the enumeration is exhausted. Another interesting implementation detail (shown in Figure 6), is that the only requirement for decryption function to be called is that the size of the value data is 0x200 (512) bytes in size. This is not as inefficient as it may first seem as the decryption function will exit relatively quickly if the first stage dropper has not performed its setup operations. Nevertheless it's clear that for the malware authors, obfuscation is more important than efficiency.

The data in the registry ontain the decryption key, but it does contain pted using the Microsoft CNG library f and stored it in the one of the system defau der". If the first stage dropper has not run, th xit. Provided that the storage provider actual ecrypted data contains a header that contains cription of the header can be found in Table 1

Offset	Descriptio
0x00	Offset to s
0x08	Size of sec
0x10	Offset to I
0x18	IV size
0x20	Offset to A
0x28	Encrypted

Now the header has be ad is encrypted using the AES algorithm. First a chunk of data from the registry is passed to the BCryptGenerateSymmetricKey function, which results in the AES decryption key being created. Once the key has been generated and the decryption properties have been set, the payload will be decrypted. The decrypted payload is then checked to ensure that it's a valid PE file (it checks for the MZ PE magic bytes, and also checks for the machine architecture entry in the PE header). If the checks pass, the file is manually loaded (imports and relocations) and the entry point is called (as shown in Figure 7).

```
memcpy(v3, v4, v5);
memcpy(v3, v8, v9);
v12 = aes_decryption(
        (__int64)v8 + *v8,        // secret key data
        v8[1],                    // secret key data size
        (__int64)v8 + v8[2],      // IV
        v8[3],                    // IV size
        (__int64)v3 + (unsigned int)v3[4],// encrypted data
        v3[5],                    // encrypted data size
        (__int64)&p_decrypted_data,
        (__int64)&decrypted_data_size);
free(Src);
Src = 0i64;
if ( !v12
  && !(unsigned int)is_pe_file(
                      (__int64)p_decrypted_data + *(unsigned int *)p_decrypted_data,
                      *((unsigned int *)p_decrypted_data + 1)) )
{
  v18 = (char *)p_decrypted_data + *((unsigned int *)p_decrypted_data + 2);
  v19 = *((_DWORD *)p_decrypted_data + 3);
  v20 = (char *)p_decrypted_data + *((unsigned int *)p_decrypted_data + 4);
  v21 = *((_DWORD *)p_decrypted_data + 5);
  v6 = load_and_exec_pe((char *)p_decrypted_data + *(unsigned int *)p_decrypted_data, 0i64, &v18);
  free(p_decrypted_data);
  p_decrypted_data = 0i64;
  if ( !v6 )
    break;
}
```

Figure 7

Summary

In this blog post we hav ... oup is now using it with a new component: Regl ... d runs it. It seems that the group is taking idea ... t is leaving as little information as possible ... he encrypted data. This means that that it is no

Thankfully all is not lost ... elow.

As part of our research ... decided to release this tool just in case others ... Defence/tree/master/S

Yara Rules

```
rule turla_png_dropper {
    meta:
        author = "Ben Humphrey"
        description = "Detects the PNG Dropper used by the Turla group"
        sha256 =
"6ed939f59476fd31dc4d99e96136e928fbd88aec0d9c59846092c0e93a3c0e27"

    strings:
        $api0 = "GdiplusStartup"
        $api1 = "GdipAlloc"
        $api2 = "GdipCreateBitmapFromStreamICM"
        $api3 = "GdipBitmapLockBits"
        $api4 = "GdipGetImageWidth"
        $api5 = "GdipGetImageHeight"
        $api6 = "GdiplusShutdown"

        $code32 = {
        8B 46 3C                // mov     eax, [esi+3Ch]
        B9 0B 01 0
        66 39 4C 3
        8B 44 30 2
        6A 00
        B9 AF BE A
        51
        51
        03 C6
        56
        FF D0
    }

    $code64 = {
        48 63 43 3
        B9 0B 01 0
        BA AF BE A
        66 39 4C 1
        8B 44 18 2
        45 33 C9
        44 8B C2
        48 8B CB
        48 03 C3
        FF D0
    }

    condition:
        (uint16(0) == 0x5A4D and uint16(uint32(0x3c)) == 0x4550) and
        all of ($api*) and
        1 of ($code*)
}
```

```
rule turla_png_reg_enum_payload {
      meta:
                author = "Ben Humphrey"
                description = "Payload that has most recently been dropped by the
Turla PNG Dropper"
               shas256 =
"fea27eb2e939e930c8617dcf64366d1649988f30555f6ee9cd09fe54e4bc22b3"

      strings:
          $crypt00 = "Microsoft Software Key Storage Provider" wide
          $crypt01 = "ChainingModeCBC" wide
          $crypt02 = "AES" wide

      condition:
          (uint16(0) == 0x5A4D and uint16(uint32(0x3c)) == 0x4550) and
          pe.imports("advapi32.dll", "StartServiceCtrlDispatcherA") and
          pe.imports("advapi32.dll", "RegEnumValueA") and
          pe.imports("advapi32.dll", "RegEnumKeyExA") and
          pe.imports("
          pe.imports("
          pe.imports("
          pe.imports("
          pe.imports("
          pe.imports("
          pe.imports("
          pe.imports("
          pe.imports("
          all of them
}
```

IOCs

Sample Analyse

- 6ed939f59476fd31dc4d
- fea27eb2e939e930c861 NG dropper)

Services

- WerFaultSvc

References

[1] https://www.carbonl g-dropper/

Published date: 22 Nov

Written by: Ben Humphrey

Matt Lewis