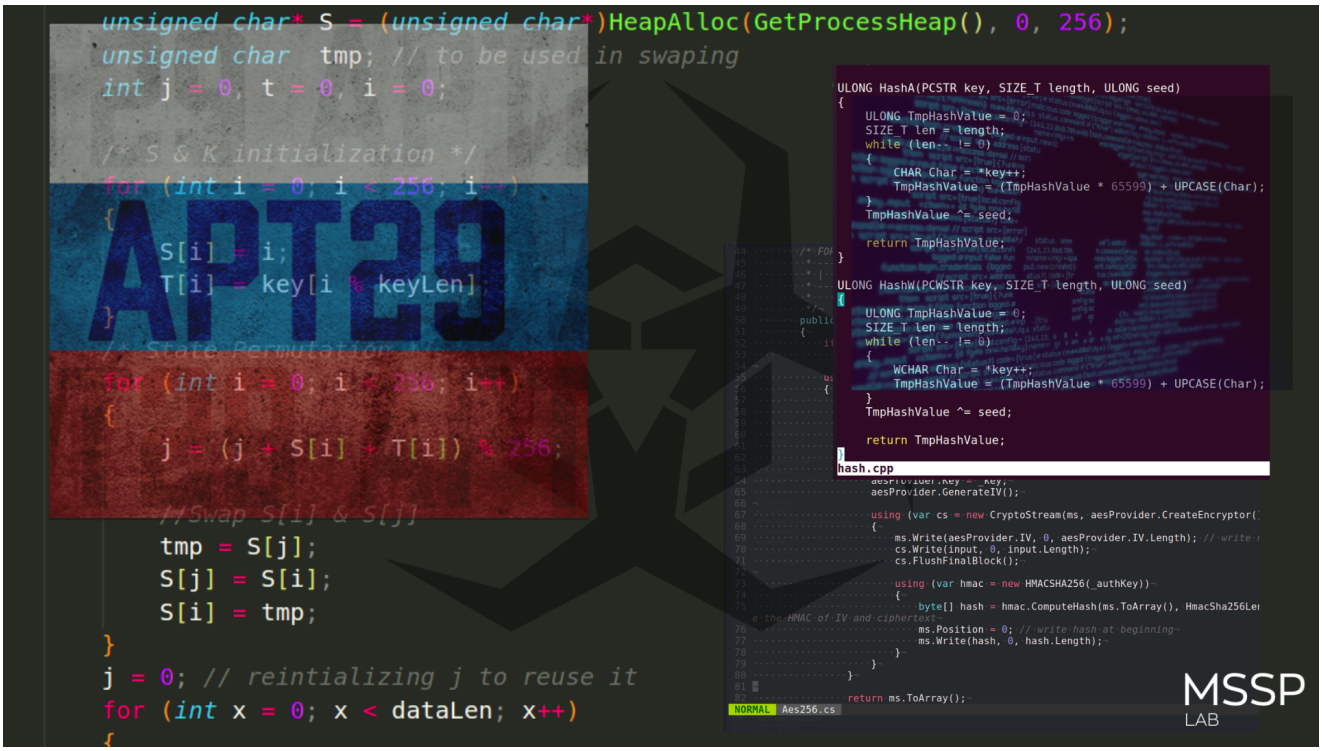
Malware analysis report: SNOWYAMBER (+APT29 related malwares)

mssplab.github.io/threat-hunting/2023/06/02/malware-analysis-apt29.html

June 2, 2023

9 minute read

This report provides a comprehensive analysis of the `SNOWYAMBER` dropper, and it's modifications, a sophisticated piece of malware attributed to the Advanced Persistent Threat group `APT29`. The group is believed to be tied to the Russian government and has been linked to numerous cyber espionage operations.



Threat actor

APT29, also known as The Dukes or Cozy Bear, is a highly sophisticated and well-resourced cyber espionage group believed to be associated with the Russian government. It has been operating since at least 2008.

Target

While the group's exact location is unknown, multiple cybersecurity research groups and government agencies attribute APT29 to Russia. Their targets are typically spread across the globe, with a specific focus on government organizations, think-tanks, healthcare organizations, and energy sectors.

TTPs

APT29 is known for its persistent and evolving tactics, which include a combination of advanced techniques and procedures:

- *Spear-Phishing:* APT29 commonly utilizes spear-phishing campaigns for initial compromise. These usually involve emails with either a malicious attachment or a link to a malicious website. Their spear-phishing attacks often involve the use of legitimate web services, such as Google accounts, to host their payload and seem less suspicious.

- *Use of Zero-days and Exploits:* The group is known to use zero-day exploits as well as known vulnerabilities to infiltrate networks. They were known to exploit vulnerabilities such as `CVE-2017-11292` (Adobe Flash), `CVE-2017-8759` (.NET Framework), and `CVE-2017-0199` (Microsoft Office/WordPad).

- *Living-off-the-Land Tactics:* `APT29` frequently employs "living-off-the-land" tactics, where they use legitimate system tools and processes to hide their activities and maintain persistence. For instance, they have been known to use PowerShell for scripting, WMI for persistence, and PsExec for lateral movement.

- *Custom Malware:* The group uses a variety of custom backdoors and droppers, including but not limited to MiniDuke, CosmicDuke, OnionDuke, and CozyDuke. More recently, they have been associated with the WellMess and WellMail malware.

- *Stealth and Long-term Persistence:* APT29 is known for its stealthy operations and ability to maintain a long-term presence on infiltrated networks without detection. They often do so by limiting their activities during the working hours of the target's local time zone to mimic legitimate users and avoid raising alerts.

- *Data Exfiltration:* APT29 is known for extracting sensitive information from the infiltrated networks. They often do this very slowly and cautiously to avoid detection. The group is believed to be interested in gathering intelligence related to foreign policy, defense, international relations, and similar topics.

Malware features

Through our analysis, we have identified the following notable features of the `SNOWYAMBER` dropper:

Infection capabilities: The malware is typically introduced to the victim's machine via spear phishing, hiding in documents that prompt the user to enable macros. (High Confidence)

Capacity for self-preservation: The malware employs anti-analysis and persistence mechanisms, which include obfuscation techniques, disabling security tools, and creating Registry keys to survive reboots. (High Confidence)

Diffusion mechanism: The dropper, upon execution, deploys additional payloads on the infected machine, and may also propagate laterally within the network. (Medium Confidence)

Data exfiltration capabilities: The malware appears capable of collecting system information and sending it to a Command and Control (C2) server. (High Confidence)

C2 mechanisms: The malware uses encrypted HTTP requests for C2 communication. (High Confidence)

Identification

Among the malware samples analysed, the most interesting are following.

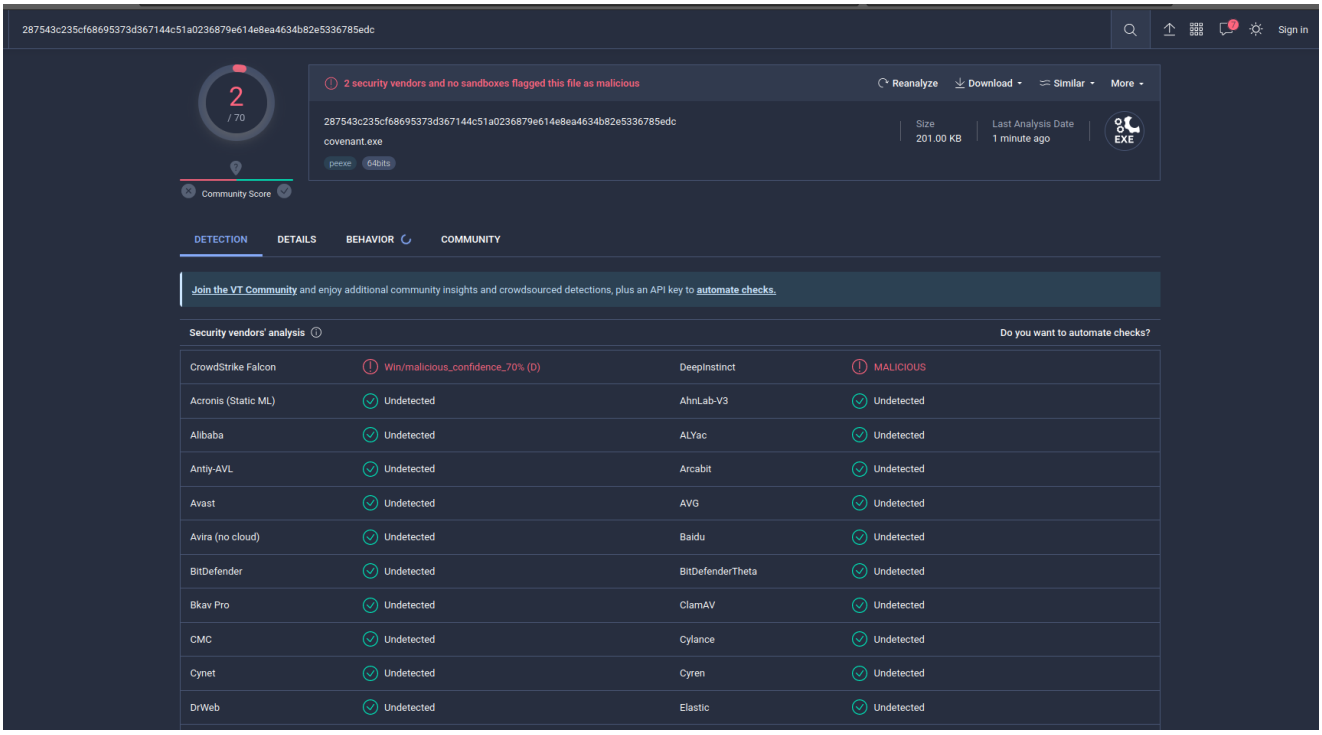
Four samples are being investigated:

sample.exe - this file is worked for injection:

File size: `205824 bytes`
MD5 sum: `109f05770bf8550f71b39ceaffc6e42e`
SHA-1 sum: `72b57b47649f145ba341420fa0a4624810c011d9`
SHA-256 sum: `287543c235cf68695373d367144c51a0236879e614e8ea4634b82e5336785edc`

First of all, check our sample via VirusTotal:

https://www.virustotal.com/gui/file/287543c235cf68695373d367144c51a0236879e614e8ea4634b82e5336785edc/detection



So, 2 of 70 AV engines detect our sample as malicious.

This sample is written in C++ and uses multiple malware development tricks: WinAPI functions call by hash, string obfuscation and encryption, time distortion.

Static analysis

The specified sample is a PE file:

```
file <sample.exe>
```

```
┌──(cocomelonc㉿kali)-[~/Desktop/shared/malware/2023-06-02-malware-analysis]
└─$ file covenant.exe
covenant.exe: PE32+ executable (GUI) x86-64, for MS Windows
```

`hexdump -C <sample.exe>`

```
┌──(cocomelonc㉿kali)-[~/Desktop/shared/malware/2023-06-02-malware-analysis]
└─$ hexdump -C ./covenant.exe | head -n 100
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 10 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  c0 f3 36 74 84 92 58 27  84 92 58 27 84 92 58 27  |..6t..X'..X'..X'|
00000090  90 f9 5c 26 8f 92 58 27  90 f9 5b 26 81 92 58 27  |..\&..X'..[&..X'|
000000a0  90 f9 5d 26 0e 92 58 27  d6 e7 5d 26 a1 92 58 27  |..]&..X'..]&..X'|
000000b0  d6 e7 5c 26 94 92 58 27  d6 e7 5b 26 8c 92 58 27  |..\&..X'..[&..X'|
000000c0  90 f9 59 26 81 92 58 27  84 92 59 27 ff 92 58 27  |..Y&..X'..Y'..X'|
000000d0  47 e7 51 26 83 92 58 27  47 e7 5b 26 85 92 58 27  |G.Q&..X'G.[&..X'|
000000e0  47 e7 a7 27 85 92 58 27  84 92 cf 27 85 92 58 27  |G..'..X'...'..X'|
000000f0  47 e7 5a 26 85 92 58 27  52 69 63 68 84 92 58 27  |G.Z&..X'Rich..X'|
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Use `exiftool` for looking metadata:

`exiftool <sample.exe>`

```
└─$ exiftool covenant.exe
ExifTool Version Number         : 12.49
File Name                       : covenant.exe
Directory                       : .
File Size                       : 206 kB
File Modification Date/Time     : 2023:02:25 08:07:22+03:00
File Access Date/Time           : 2023:06:02 17:23:20+03:00
File Inode Change Date/Time     : 2023:06:02 17:18:56+03:00
File Permissions                : -rw-r--r--
File Type                       : Win64 EXE
File Type Extension             : exe
MIME Type                       : application/octet-stream
Machine Type                    : AMD AMD64
Time Stamp                      : 2023:02:25 22:07:22+03:00
Image File Characteristics      : Executable, Large address aware
PE Type                         : PE32+
Linker Version                  : 14.29
Code Size                       : 53760
Initialized Data Size           : 164352
Uninitialized Data Size         : 0
Entry Point                     : 0x40a8
OS Version                      : 6.0
Image Version                   : 0.0
Subsystem Version               : 6.0
Subsystem                       : Windows GUI
```

And we see that file timestamp is `2023-02-25 22:07:22+03.00`

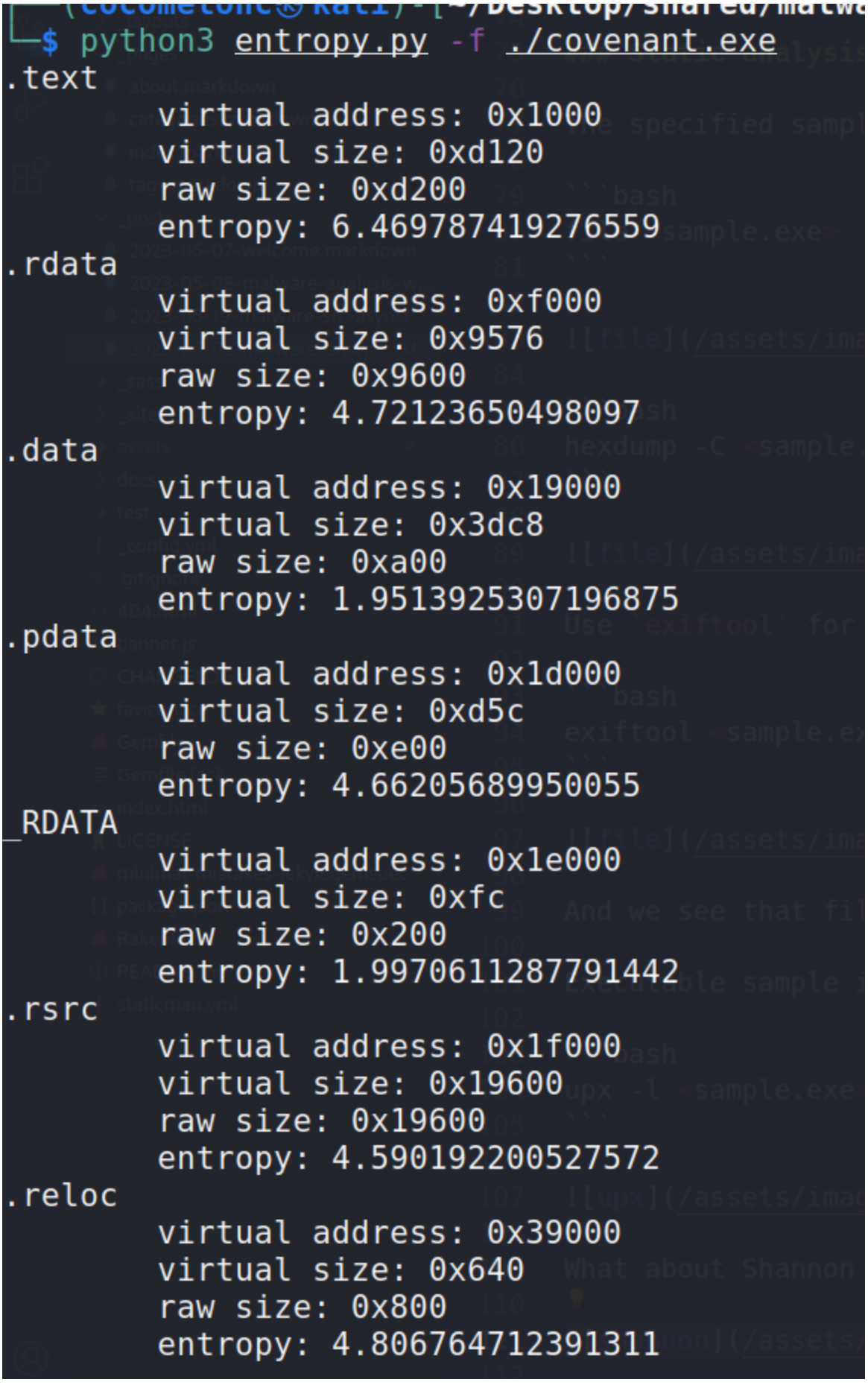
Executable sample is not packed by `upx`:

```
upx -l <sample.exe>
```

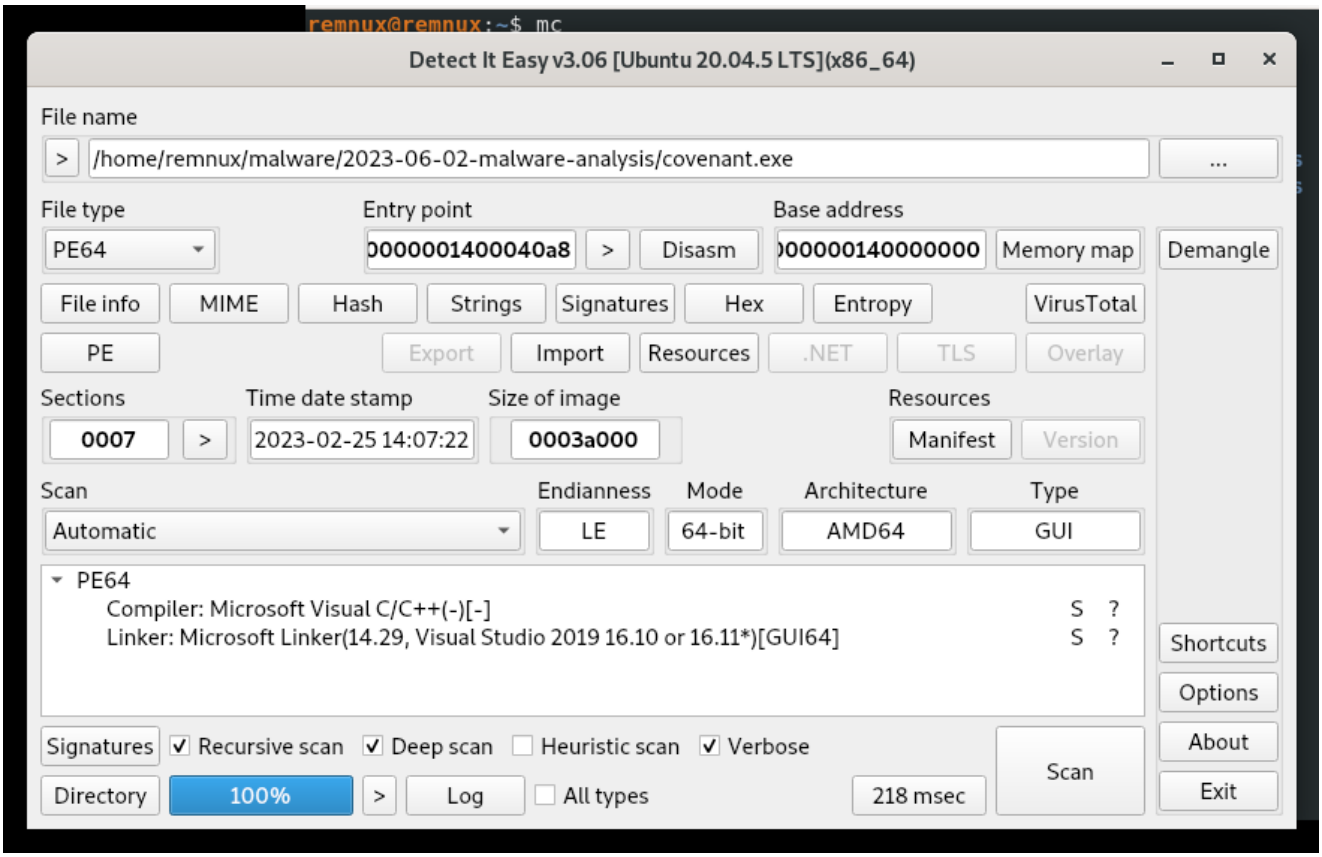
```
└─$ upx -l covenant.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96        Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
upx: covenant.exe: NotPackedException: not packed by UPX
```

What about Shannon entropy of the sample:

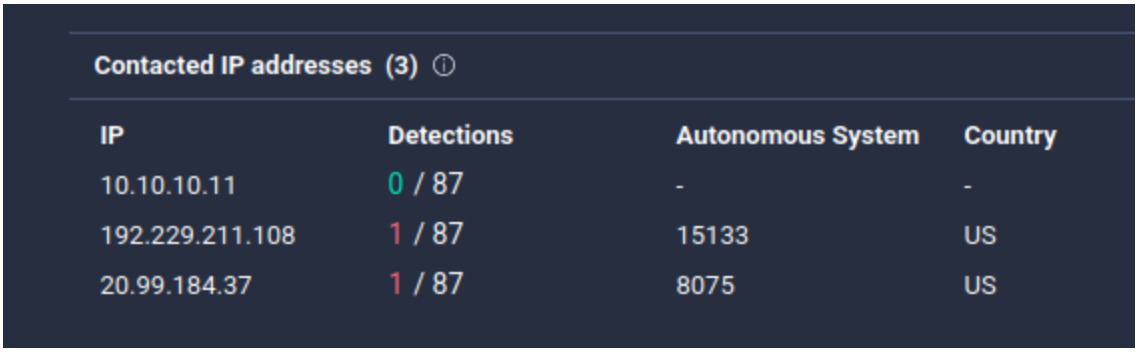

$ python3 entropy.py -f ./covenant.exe
.text
virtual address: 0x1000
virtual size: 0xd120
raw size: 0xd200
entropy: 6.469787419276559
.rdata
virtual address: 0xf000
virtual size: 0x9576
raw size: 0x9600
entropy: 4.72123650498097
.data
virtual address: 0x19000
virtual size: 0x3dc8
raw size: 0xa00
entropy: 1.9513925307196875
.pdata
virtual address: 0x1d000
virtual size: 0xd5c
raw size: 0xe00
entropy: 4.66205689950055
_RDATA
virtual address: 0x1e000
virtual size: 0xfc
raw size: 0x200
entropy: 1.9970611287791442
.rsrc
virtual address: 0x1f000
virtual size: 0x19600
raw size: 0x19600
entropy: 4.590192200527572
.reloc
virtual address: 0x39000
virtual size: 0x640
raw size: 0x800
entropy: 4.806764712391311

Analysze with DIE says that the compiler is `Microsoft Visual Studio 2019`:



dynamic analysis

Contacted IP addresses is:



The main logic starts with the `int start` function.

Then arbitrary computations are performed: this is a popular sandbox bypass trick. And run `switch` logic:

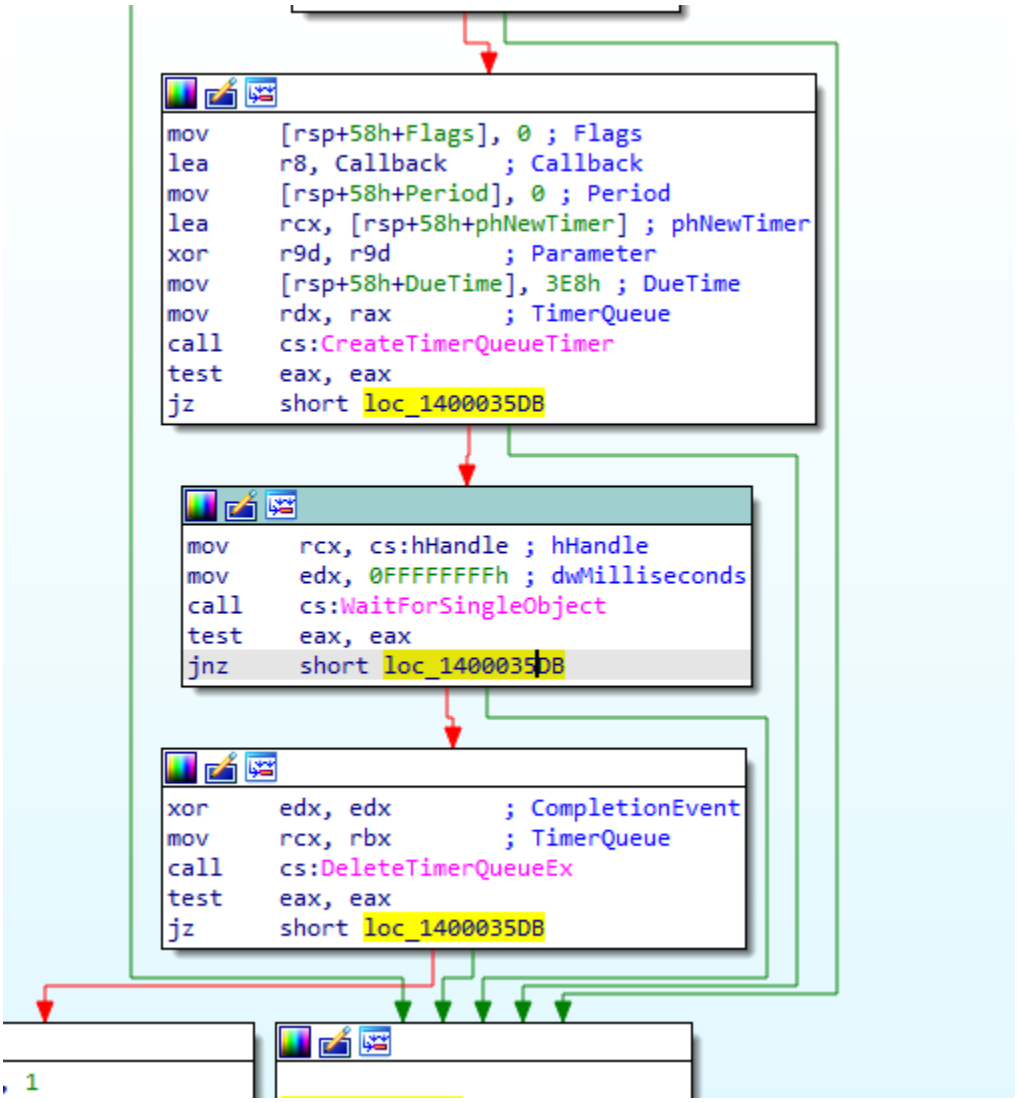
```
1
2  void FUN_1400032b0(undefined *param_1)
3
4  {
5    int iVar1;
6    uint uVar2;
7    BOOL BVar3;
8    DWORD dwMessageId;
9    HANDLE hConsoleOutput;
10   int iVar4;
11   wchar_t *lpText;
12   uint uVar5;
13   int iVar6;
14   int iVar7;
15   int iVar8;
16   int iVar9;
17   bool bVar10;
18   undefined auStackY_88 [32];
19   undefined8 local_40;
20   SMALL_RECT local_38;
21   _CONSOLE_HISTORY_INFO local_30;
22   ulonglong local_20;
23
24   local_20 = DAT_140019008 ^ (ulonglong)auStackY_88;
25   iVar7 = -1;
26   iVar8 = 0x135;
27   iVar6 = 0;
28   iVar9 = 2;
29   do {
30     iVar4 = iVar7 + 2;
31     iVar1 = (iVar8 + 7) / 2 + iVar9;
32     iVar9 = iVar7 + 1;
33     iVar8 = (iVar1 + iVar8 + 0x13) * 6;
34     bVar10 = iVar7 != 2;
35     iVar7 = 6;
36     if (bVar10) {
37       iVar7 = iVar4;
38     }
39   } while (iVar7 < 0xd);
40   uVar5 = iVar8 + 7 + iVar7 + (iVar9 - iVar8 / 2) * 2;
41   for (uVar2 = uVar5; (0 < (int)uVar2 && ((uVar2 & 1) == 0)); uVar2 = (int)uVar2 / 2) {
42     iVar6 = iVar6 + 1;
43   }
44   switch(uVar2) {
45   case 0xc:
46     MessageBoxA((HWND)0x0,"Sorry, Unkown Error Occured","Error !",0);
47     break;
```

```
.text:000000014000336D                mov     ecx, ds:(jpt_140003377 - 140000000h)[rax+rax*4]
.text:0000000140003374                add     rcx, rdx
.text:0000000140003377                jmp     rcx             ; switch jump
.text:0000000140003379 ; ---------------------------------------------------------------------------
.text:0000000140003379
.text:0000000140003379 loc_140003379:                         ; CODE XREF: sub_1400032B0+C7↑j
.text:0000000140003379                                        ; DATA XREF: sub_1400032B0:jpt_140003377↓o
.text:0000000140003379                xor     r9d, r9d        ; jumptable 0000000140003377 case 12
.text:000000014000337C                lea     r8, Caption     ; "Error !"
.text:0000000140003383                lea     rdx, Text       ; "Sorry, Unkown Error Occured"
.text:000000014000338A                xor     ecx, ecx        ; hWnd
.text:000000014000338C                call    cs:MessageBoxA
.text:0000000140003392                jmp     short def_140003377 ; jumptable 0000000140003377 default case
.text:0000000140003394 ; ---------------------------------------------------------------------------
.text:0000000140003394
.text:0000000140003394 loc_140003394:                         ; CODE XREF: sub_1400032B0+C7↑j
.text:0000000140003394                                        ; DATA XREF: sub_1400032B0:jpt_140003377↓o
.text:0000000140003394                xor     r9d, r9d        ; jumptable 0000000140003377 case 13
.text:0000000140003397                lea     r8, Caption     ; "Error !"
.text:000000014000339E                lea     rdx, aSorryPleaseIns ; "Sorry, Please Install The Software Agai"...
.text:00000001400033A5                xor     ecx, ecx        ; hWnd
.text:00000001400033A7                call    cs:MessageBoxA
.text:00000001400033AD                jmp     short def_140003377 ; jumptable 0000000140003377 default case
.text:00000001400033AF ; ---------------------------------------------------------------------------
.text:00000001400033AF
.text:00000001400033AF loc_1400033AF:                         ; CODE XREF: sub_1400032B0+C7↑j
.text:00000001400033AF                                        ; DATA XREF: sub_1400032B0:jpt_140003377↓o
.text:00000001400033AF                xor     r9d, r9d        ; jumptable 0000000140003377 case 14
.text:00000001400033B2                lea     r8, Caption     ; "Error !"
.text:00000001400033B9                lea     rdx, aSorrySomeLibra ; "Sorry, Some Libraries Are Missing"
.text:00000001400033C0                xor     ecx, ecx        ; hWnd
.text:00000001400033C2                call    cs:MessageBoxA
.text:00000001400033C8                jmp     short def_140003377 ; jumptable 0000000140003377 default case
.text:00000001400033CA ; ---------------------------------------------------------------------------
00002779 0000000140003379: sub_1400032B0:loc_140003379 (Synchronized with Hex View-1)
```

Also use WinAPI functions:

- `CreateTimerQueue`
- `CreateTimerQueueTimer`
- `DeleteTimerQueueEx`

```
.text:0000000140003541                xor     r8d, r8d        ; bInitialState
.text:0000000140003544                xor     ecx, ecx        ; lpEventAttributes
.text:0000000140003546                lea     edx, [r9+1]     ; bManualReset
.text:000000014000354A                call    cs:CreateEventW
.text:0000000140003550                mov     cs:hHandle, rax
.text:0000000140003557                test    rax, rax
.text:000000014000355A                jz      short loc_1400035DB
.text:000000014000355C                call    cs:CreateTimerQueue
.text:0000000140003562                mov     rbx, rax
.text:0000000140003565                test    rax, rax
.text:0000000140003568                jz      short loc_1400035DB
.text:000000014000356A                mov     [rsp+58h+Flags], 0 ; Flags
.text:0000000140003572                lea     r8, Callback    ; Callback
.text:0000000140003579                mov     [rsp+58h+Period], 0 ; Period
.text:0000000140003581                lea     rcx, [rsp+58h+phNewTimer] ; phNewTimer
.text:0000000140003586                xor     r9d, r9d        ; Parameter
.text:0000000140003589                mov     [rsp+58h+DueTime], 3E8h ; DueTime
.text:0000000140003591                mov     rdx, rax        ; TimerQueue
.text:0000000140003594                call    cs:CreateTimerQueueTimer
.text:000000014000359A                test    eax, eax
.text:000000014000359C                jz      short loc_1400035DB
.text:000000014000359E                mov     rcx, cs:hHandle ; hHandle
.text:00000001400035A5                mov     edx, 0FFFFFFFFh ; dwMilliseconds
.text:00000001400035AA                call    cs:WaitForSingleObject
.text:00000001400035B0                test    eax, eax
.text:00000001400035B2                jnz     short loc_1400035DB
.text:00000001400035B4                xor     edx, edx        ; CompletionEvent
.text:00000001400035B6                mov     rcx, rbx        ; TimerQueue
.text:00000001400035B9                call    cs:DeleteTimerQueueEx
.text:00000001400035BF                test    eax, eax
.text:00000001400035C1                jz      short loc_1400035DB
.text:00000001400035C3                mov     eax, 1
.text:00000001400035C8                mov     rcx, [rsp+58h+var_10]
.text:00000001400035CD                xor     rcx, rsp
.text:00000001400035D0                call    sub_140003E30
.text:00000001400035D5                add     rsp, 50h
.text:00000001400035D9                pop     rbx
.text:00000001400035DA                retn
```



Here use an event object to track the TimeRoutine execution, create the timer queue, then set a timer to call the timer routine in `10` seconds.

This implementation sets up asynchronous timers using `CreateTimerQueueTimer`. Each executes one after the other and does the following tasks: Wait a specific time period.

```
void FUN_140003520(void)

{
  BOOL BVar1;
  DWORD DVar2;
  HANDLE TimerQueue;
  undefined auStackY_58 [32];
  HANDLE local_18;
  ulonglong local_10;

  local_10 = DAT_140019008 ^ (ulonglong)auStackY_58;
  local_18 = (HANDLE)0x0;
  DAT_1401ad98 = CreateEventW((LPSECURITY_ATTRIBUTES)0x0,1,0,(LPCWSTR)0x0);
  if (DAT_1401ad98 != (HANDLE)0x0) {
    TimerQueue = CreateTimerQueue();
    if (TimerQueue != (HANDLE)0x0) {
      BVar1 = CreateTimerQueueTimer(&local_18,TimerQueue,FUN_1400034f0,(PVOID)0x0,1000,0,0);
      if (BVar1 != 0) {
        DVar2 = WaitForSingleObject(DAT_1401ad98,0xffffffff);
        if (DVar2 == 0) {
          BVar1 = DeleteTimerQueueEx(TimerQueue,(HANDLE)0x0);
          if (BVar1 != 0) {
            FUN_140003e30(local_10 ^ (ulonglong)auStackY_58);
            return;
          }
        }
      }
    }
  }
  FUN_140003e30(local_10 ^ (ulonglong)auStackY_58);
  return;
}
```

When executed, the injector reads the resource, decrypts it by RC4 algorithm, allocates memory, copies sections, processes relocks, and transfers control to the entry point.

What about injection technique. It's PE injection.

```
PIMAGE_SECTION_HEADER __cdecl _FindPESection(PBYTE pImageBase,DWORD_PTR rva)

{
  int iVar1;
  PIMAGE_SECTION_HEADER p_Var2;
  uint uVar3;

  iVar1 = *(int *)(pImageBase + 0x3c);
  uVar3 = 0;
  p_Var2 = (PIMAGE_SECTION_HEADER)
           (pImageBase +
           (ulonglong)*(ushort *)(pImageBase + (longlong)iVar1 + 0x14) + 0x18 + (longlong)iVar1);
  if (*(ushort *)(pImageBase + (longlong)iVar1 + 6) != 0) {
    do {
      if ((p_Var2->VirtualAddress <= rva) &&
         (rva < (p_Var2->Misc).PhysicalAddress + p_Var2->VirtualAddress)) {
        return p_Var2;
      }
      uVar3 = uVar3 + 1;
      p_Var2 = p_Var2 + 1;
    } while (uVar3 < *(ushort *)(pImageBase + (longlong)iVar1 + 6));
  }
  return (PIMAGE_SECTION_HEADER)0x0;
}
```

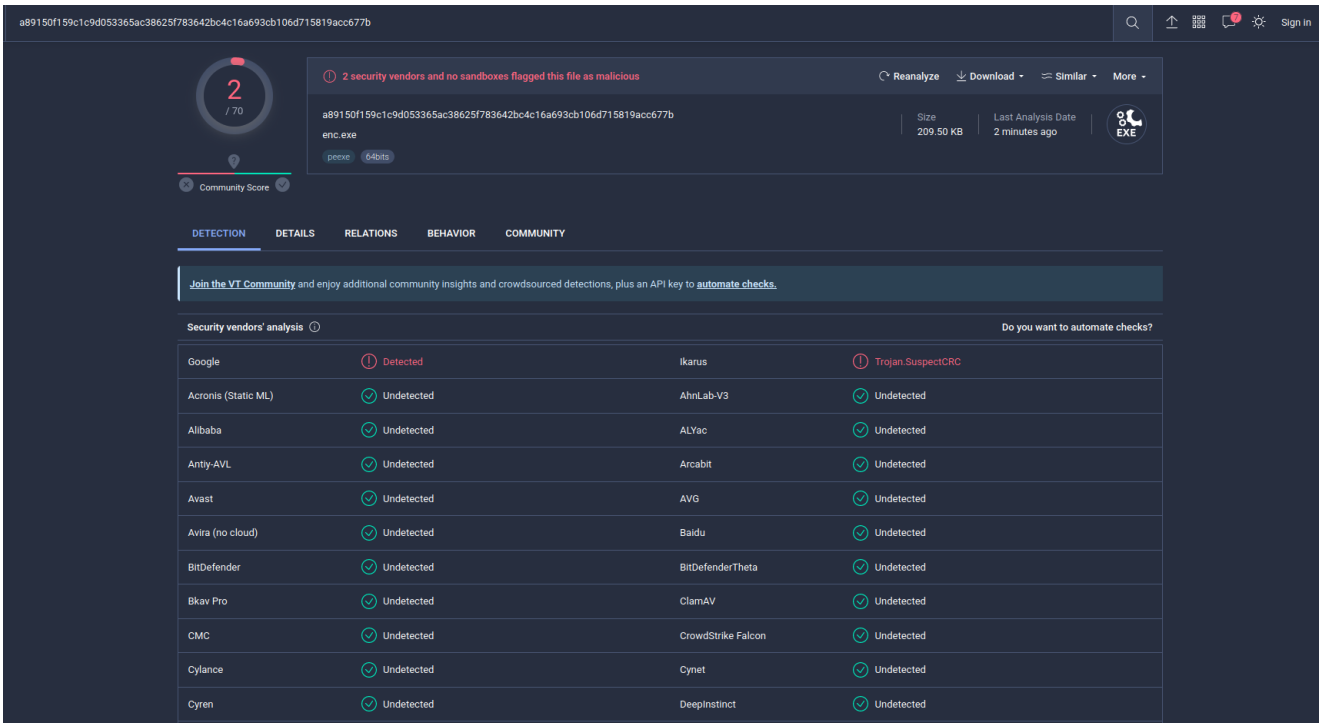
All NT API functions are replaced by calling equivalent syscalls from https://github.com/klezVirus/SysWhispers3.

sample2.exe - this sample is an encryptor:

File size: `214528 bytes`
MD5 sum: `107dae5b9c61c962e0d604cd70a1d8ae`
SHA-1 sum: `3752be6b162bacb0d7c12b6d122c9dbaf3ad6223`
SHA-256 sum: `a89150f159c1c9d053365ac38625f783642bc4c16a693cb106d715819acc677b`

Check it via VirusTotal:

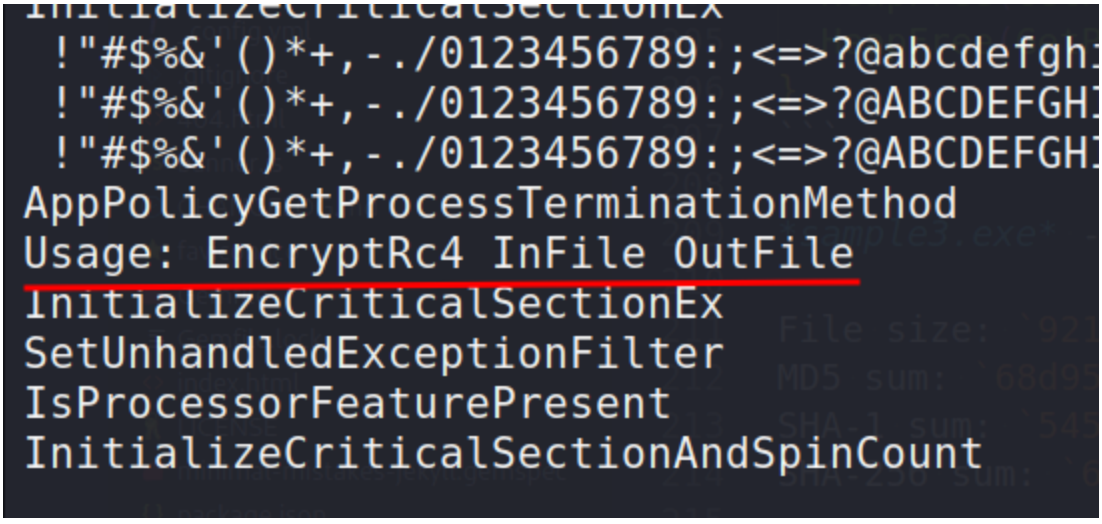
https://www.virustotal.com/gui/file/a89150f159c1c9d053365ac38625f783642bc4c16a693cb106d715819acc677b/detection

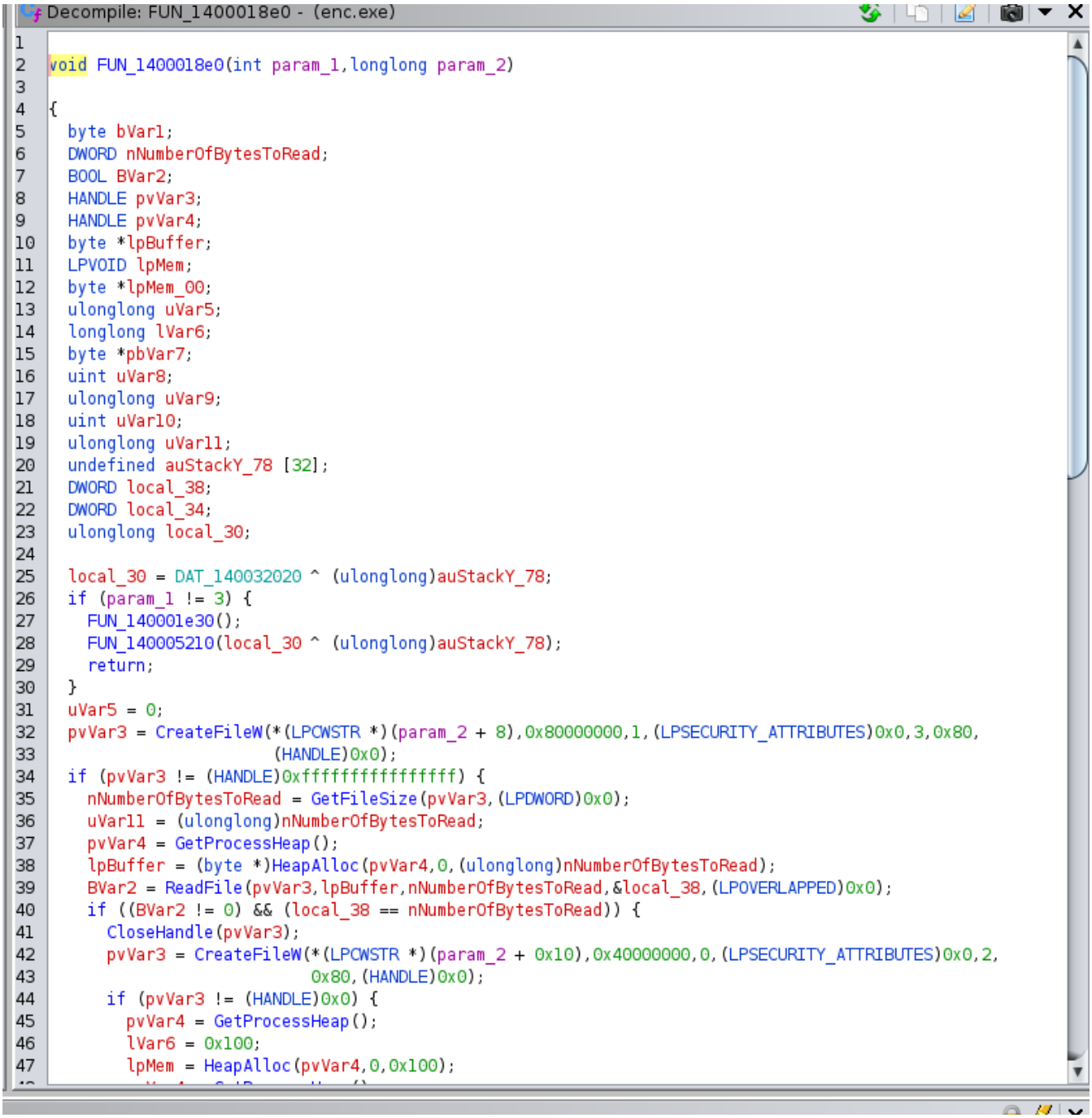


So, 2 of 70 AV engines detect our sample as malicious.

This encryptor encrypts the payload with the RC4 algorithm, then the result is attached to the injector with the resource.

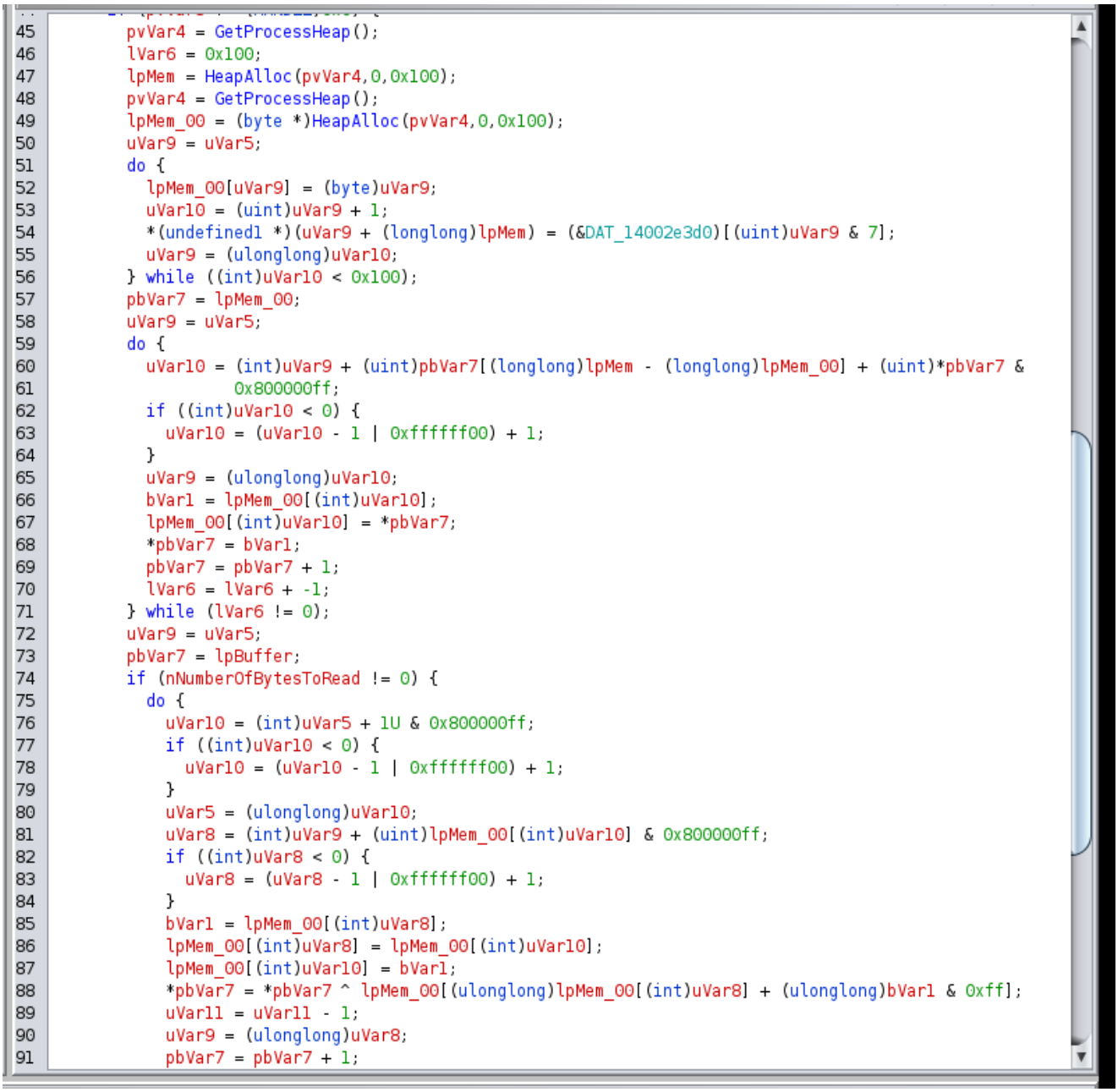
Encryptor use 2 params: Input file and output file.






```
1
2   void FUN_1400018e0(int param_1,longlong param_2)
3
4   {
5     byte bVar1;
6     DWORD nNumberOfBytesToRead;
7     BOOL BVar2;
8     HANDLE pvVar3;
9     HANDLE pvVar4;
10    byte *lpBuffer;
11    LPVOID lpMem;
12    byte *lpMem_00;
13    ulonglong uVar5;
14    longlong lVar6;
15    byte *pbVar7;
16    uint uVar8;
17    ulonglong uVar9;
18    uint uVar10;
19    ulonglong uVar11;
20    undefined auStackY_78 [32];
21    DWORD local_38;
22    DWORD local_34;
23    ulonglong local_30;
24
25    local_30 = DAT_140032020 ^ (ulonglong)auStackY_78;
26    if (param_1 != 3) {
27      FUN_140001e30();
28      FUN_140005210(local_30 ^ (ulonglong)auStackY_78);
29      return;
30    }
31    uVar5 = 0;
32    pvVar3 = CreateFileW(*(LPCWSTR *)(param_2 + 8),0x80000000,1,(LPSECURITY_ATTRIBUTES)0x0,3,0x80,
33                         (HANDLE)0x0);
34    if (pvVar3 != (HANDLE)0xffffffffffffffff) {
35      nNumberOfBytesToRead = GetFileSize(pvVar3,(LPDWORD)0x0);
36      uVar11 = (ulonglong)nNumberOfBytesToRead;
37      pvVar4 = GetProcessHeap();
38      lpBuffer = (byte *)HeapAlloc(pvVar4,0,(ulonglong)nNumberOfBytesToRead);
39      BVar2 = ReadFile(pvVar3,lpBuffer,nNumberOfBytesToRead,&local_38,(LPOVERLAPPED)0x0);
40      if ((BVar2 != 0) && (local_38 == nNumberOfBytesToRead)) {
41        CloseHandle(pvVar3);
42        pvVar3 = CreateFileW(*(LPCWSTR *)(param_2 + 0x10),0x40000000,0,(LPSECURITY_ATTRIBUTES)0x0,2,
43                             0x80,(HANDLE)0x0);
44        if (pvVar3 != (HANDLE)0x0) {
45          pvVar4 = GetProcessHeap();
46          lVar6 = 0x100;
47          lpMem = HeapAlloc(pvVar4,0,0x100);
```

Classic RC4 algorithm:



```
45        pvVar4 = GetProcessHeap();
46        lVar6 = 0x100;
47        lpMem = HeapAlloc(pvVar4,0,0x100);
48        pvVar4 = GetProcessHeap();
49        lpMem_00 = (byte *)HeapAlloc(pvVar4,0,0x100);
50        uVar9 = uVar5;
51        do {
52          lpMem_00[uVar9] = (byte)uVar9;
53          uVar10 = (uint)uVar9 + 1;
54          *(undefined1 *)(uVar9 + (longlong)lpMem) = (&DAT_14002e3d0)[(uint)uVar9 & 7];
55          uVar9 = (ulonglong)uVar10;
56        } while ((int)uVar10 < 0x100);
57        pbVar7 = lpMem_00;
58        uVar9 = uVar5;
59        do {
60          uVar10 = (int)uVar9 + (uint)pbVar7[(longlong)lpMem - (longlong)lpMem_00] + (uint)*pbVar7 &
61                   0x800000ff;
62          if ((int)uVar10 < 0) {
63            uVar10 = (uVar10 - 1 | 0xffffff00) + 1;
64          }
65          uVar9 = (ulonglong)uVar10;
66          bVar1 = lpMem_00[(int)uVar10];
67          lpMem_00[(int)uVar10] = *pbVar7;
68          *pbVar7 = bVar1;
69          pbVar7 = pbVar7 + 1;
70          lVar6 = lVar6 + -1;
71        } while (lVar6 != 0);
72        uVar9 = uVar5;
73        pbVar7 = lpBuffer;
74        if (nNumberOfBytesToRead != 0) {
75          do {
76            uVar10 = (int)uVar5 + 1U & 0x800000ff;
77            if ((int)uVar10 < 0) {
78              uVar10 = (uVar10 - 1 | 0xffffff00) + 1;
79            }
80            uVar5 = (ulonglong)uVar10;
81            uVar8 = (int)uVar9 + (uint)lpMem_00[(int)uVar10] & 0x800000ff;
82            if ((int)uVar8 < 0) {
83              uVar8 = (uVar8 - 1 | 0xffffff00) + 1;
84            }
85            bVar1 = lpMem_00[(int)uVar8];
86            lpMem_00[(int)uVar8] = lpMem_00[(int)uVar10];
87            lpMem_00[(int)uVar10] = bVar1;
88            *pbVar7 = *pbVar7 ^ lpMem_00[(ulonglong)lpMem_00[(int)uVar8] + (ulonglong)bVar1 & 0xff];
89            uVar11 = uVar11 - 1;
90            uVar9 = (ulonglong)uVar8;
91            pbVar7 = pbVar7 + 1;
```

There is a simple reimplementation this logic:

```
VOID rc4crypt(PBYTE data, PCSTR key, UINT keyLen, UINT dataLen) {
  unsigned char* T = (unsigned char*)HeapAlloc(GetProcessHeap(), 0, 256);
  unsigned char* S = (unsigned char*)HeapAlloc(GetProcessHeap(), 0, 256);
  unsigned char  tmp; // to be used in swaping
  int j = 0, t = 0, i = 0;

  /* S & K initialization */
  for (int i = 0; i < 256; i++) {
    S[i] = i;
    T[i] = key[i % keyLen];
  }

  /* State Permutation */
  for (int i = 0; i < 256; i++) {
    j = (j + S[i] + T[i]) % 256;

    //Swap S[i] & S[j]
    tmp = S[j];
    S[j] = S[i];
    S[i] = tmp;
  }
  j = 0; // reintializing j to reuse it
  for (int x = 0; x < dataLen; x++) {
    i = (i + 1) % 256; // using %256 to avoid exceed the array limit
    j = (j + S[i]) % 256; // using %256 to avoid exceed the array limit

    //Swap S[i] & S[j]
    tmp = S[j];
    S[j] = S[i];
    S[i] = tmp;

    t = (S[i] + S[j]) % 256;

    data[x] = data[x] ^ S[t]; // XOR generated S[t] with Byte from the plaintext /
cipher and append each Encrypted/Decrypted byte to result array
  }

  HeapFree(GetProcessHeap(), 0, T);
  HeapFree(GetProcessHeap(), 0, S);
}
```

The encryption/decryption key is:

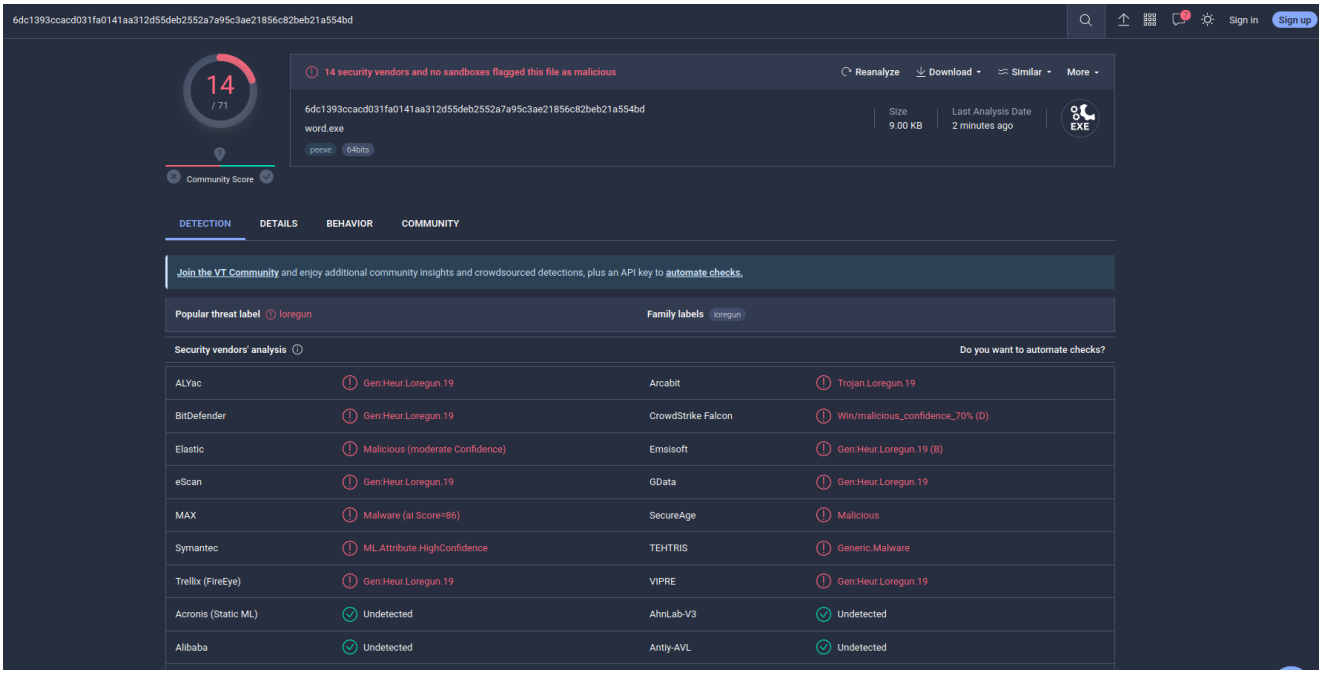
```
PCSTR key = "C2B55923\0";
```

sample3.exe - this sample plays the role of a reverse shell:

File size: `9216 bytes`
MD5 sum: `68d957f5fbb2f2078da9059995ece969`
SHA-1 sum: `545ccdb7e68c6cef6271698c0815db33625aae03`
SHA-256 sum: `6dc1393ccacd031fa0141aa312d55deb2552a7a95c3ae21856c82beb21a554bd`

First of all, check our sample via VirusTotal:

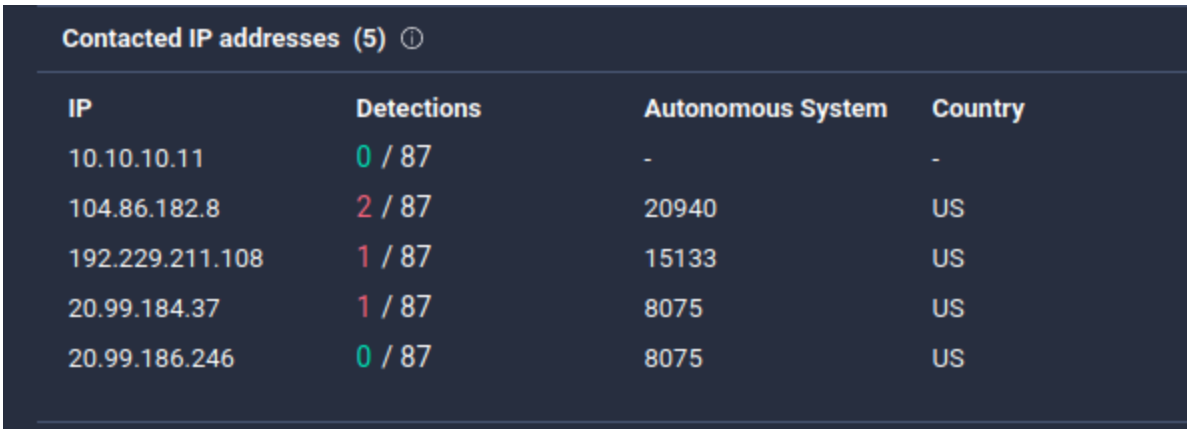
https://www.virustotal.com/gui/file/6dc1393ccacd031fa0141aa312d55deb2552a7a95c3ae21856c82beb21a554bd/detection



So, 14 of 71 AV engines detect our sample as malicious.

More of them detect file as `Gen:Heur.Loregun.19`.

Contacted IP addresses:



The logic of this sample is pretty simple: create a socket, listen on it, transfer all I/O to the socket.

Malware evasion tricks

RVA to offset:

```
6
7 PIMAGE_SECTION_HEADER __cdecl _FindPESection(PBYTE pImageBase,DWORD_PTR rva)
8
9 {
10   int iVar1;
11   PIMAGE_SECTION_HEADER p_Var2;
12   uint uVar3;
13
14   iVar1 = *(int *)(pImageBase + 0x3c);
15   uVar3 = 0;
16   p_Var2 = (PIMAGE_SECTION_HEADER)
17            (pImageBase +
18            (ulonglong)*(ushort *)(pImageBase + (longlong)iVar1 + 0x14) + 0x18 + (longlong)iVar1);
19   if (*(ushort *)(pImageBase + (longlong)iVar1 + 6) != 0) {
20     do {
21       if ((p_Var2->VirtualAddress <= rva) &&
22          (rva < (p_Var2->Misc).PhysicalAddress + p_Var2->VirtualAddress)) {
23         return p_Var2;
24       }
25       uVar3 = uVar3 + 1;
26       p_Var2 = p_Var2 + 1;
27     } while (uVar3 < *(ushort *)(pImageBase + (longlong)iVar1 + 6));
28   }
29   return (PIMAGE_SECTION_HEADER)0x0;
30 }
31
```

We restored WinAPI hashing logic:

```
#define UPCASE(wch)                       \
  (((wch) >= 'a') && ((wch) <= 'z') ? \
      (wch)                           \
  :                                   \
      ((wch) + ('a'-'A'))             \
  )

ULONG HashA(PCSTR key, SIZE_T length, ULONG seed) {
  ULONG TmpHashValue = 0;
  SIZE_T len = length;
  while (len-- != 0) {
    CHAR Char = *key++;
    TmpHashValue = (TmpHashValue * 65599) + UPCASE(Char);
  }
  TmpHashValue ^= seed;
  return TmpHashValue;
}

ULONG HashW(PCWSTR key, SIZE_T length, ULONG seed) {
  ULONG TmpHashValue = 0;
  SIZE_T len = length;
  while (len-- != 0) {
    WCHAR Char = *key++;
    TmpHashValue = (TmpHashValue * 65599) + UPCASE(Char);
  }
  TmpHashValue ^= seed;
  return TmpHashValue;
}
```

and hashing table:

```
122           if ((uVar9 ^ 0x59a59b2c) == uVar8) {
123             lVar12 = plVar14[4];
124             if (lVar12 != 0) {
125               local_3d8[0] = 0x8ba508f3;
126               local_3d8[1] = 0xc4f4eb06;
127               local_3d8[2] = 0xea48872c;
128               local_3d8[3] = 0x3bc77547;
129               local_3c8 = 0x96d3ba46;
130               local_3c4 = 0x8278b698;
131               local_3c0 = 0x609502e8;
132               local_3bc = 0x1347fdfd;
133               local_3b8 = 0x27bcb342;
134               local_3b4 = 0x5366ed60;
135               local_3b0 = 0x39c8604e;
136               local_3ac = 0x26f75d64;
137               local_3a8 = 0x55e2cac2;
138               local_3a4 = 0xab26d610;
139               local_3a0 = 0xbd71d0e0;
140               local_39c = 0xbd71d0ce;
141               local_398 = 0x23dcad1c;
142               local_394 = 0xd84484d6;
143               local_390 = 0x23dcad6a;
144               local_38c = 0xdfc07835;
145               local_388 = 0xdfc07803;
146               local_384 = 0xd844bb3c;
147               local_380 = 0x468d52ab;
148               local_37c = 0x468d525d;
149               local_378 = 0xf94e8b9f;
150               local_374 = 0x326d0bc9;
151               local_370 = 0x11983657;
152               local_36c = 0x9293ab58;
153               local_368 = 0xc83db0b4;
154               local_364 = 0xbdcac89f;
155               local_360 = 0x3c3e5b30;
156               local_35c = 0x1dc948b0;
157               local_358 = 0x3519f2b9;
158               local_354 = 0xce521091;
159               local_350 = 0x4ead0e2e;
160               local_34c = 0xfc4d07c0;
161               local_348 = 0xe24742d8;
162               local_344 = 0xb1d4a311;
163               local_340 = 0x819c55ff;
164               local_33c = 0xd66bb51c;
```

```
0x8ba508f3,//AmsiScanBuffer
0xc4f4eb06,//AmsiOpenSession
0xea48872c,//CloseHandle
0x3bc77547,//closesocket
0x96d3ba46,//connect
0x8278b698,//CreateMutexW
0x609502e8,//CreateProcessW
0x1347fdfd,//ExitProcess
0x27bcb342,//ExpandEnvironmentStringsW
0x5366ed60,//FreeAddrInfoW
0x39c8604e,//GetAddrInfoW
0x26f75d64,//GetCurrentThreadId
0x55e2cac2,//GetFileAttributesW
0xab26d610,//GetLastError
0xbd71d0e0,//LoadLibraryA
0xbd71d0ce,//LoadLibraryW
0x23dcad1c,//lstrcatA
0xd84484d6,//lstrcpyA
0x23dcad6a,//lstrcatW
0xdfc07835,//lstrcmpiA
0xdfc07803,//lstrcmpiW
0xd844bb3c,//lstrcpyW
0x468d52ab,//lstrlenW
0x468d525d,//lstrlenA
0xf94e8b9f,//MessageBoxW
0x326d0bc9,//MultiByteToWideChar
0x11983657,//NtTraceEvent
0x9293ab58,//OutputDebugStringW
0xc83db0b4,//ReleaseMutex
0xbdcac89f,//RtlAllocateHeap
0x3c3e5b30,//RtlCompareMemory
0x1dc948b0,//RtlMoveMemory
0x3519f2b9,//RtlDosPathNameToNtPathName_U
0xce521091,//RtlExitUserThread
0x4ead0e2e,//RtlFreeHeap
0xfc4d07c0,//RtlGetVersion
0xe24742d8,//RtlInitUnicodeString
0xb1d4a311,//RtlNtStatusToDosError
0x819c55ff,//RtlZeroMemory
0xd66bb51c,//SetLastError
0x0f8a6e1b,//Sleep
0x036a4566,//VirtualAlloc
0x0033e9b1,//VirtualAllocEx
0xc7433c7b,//VirtualFree
0xaa9a1e06,//VirtualFreeEx
0x61462271,//VirtualQuery
0x9f79559c,//WaitForMultipleObjects
0x4b570e37,//WaitForSingleObject
0x85729171,//WideCharToMultiByte
0x874700d3,//WSACleanup
0x90b71e53,//WSASocketW
0xa48ed094,//WSAStartup
```

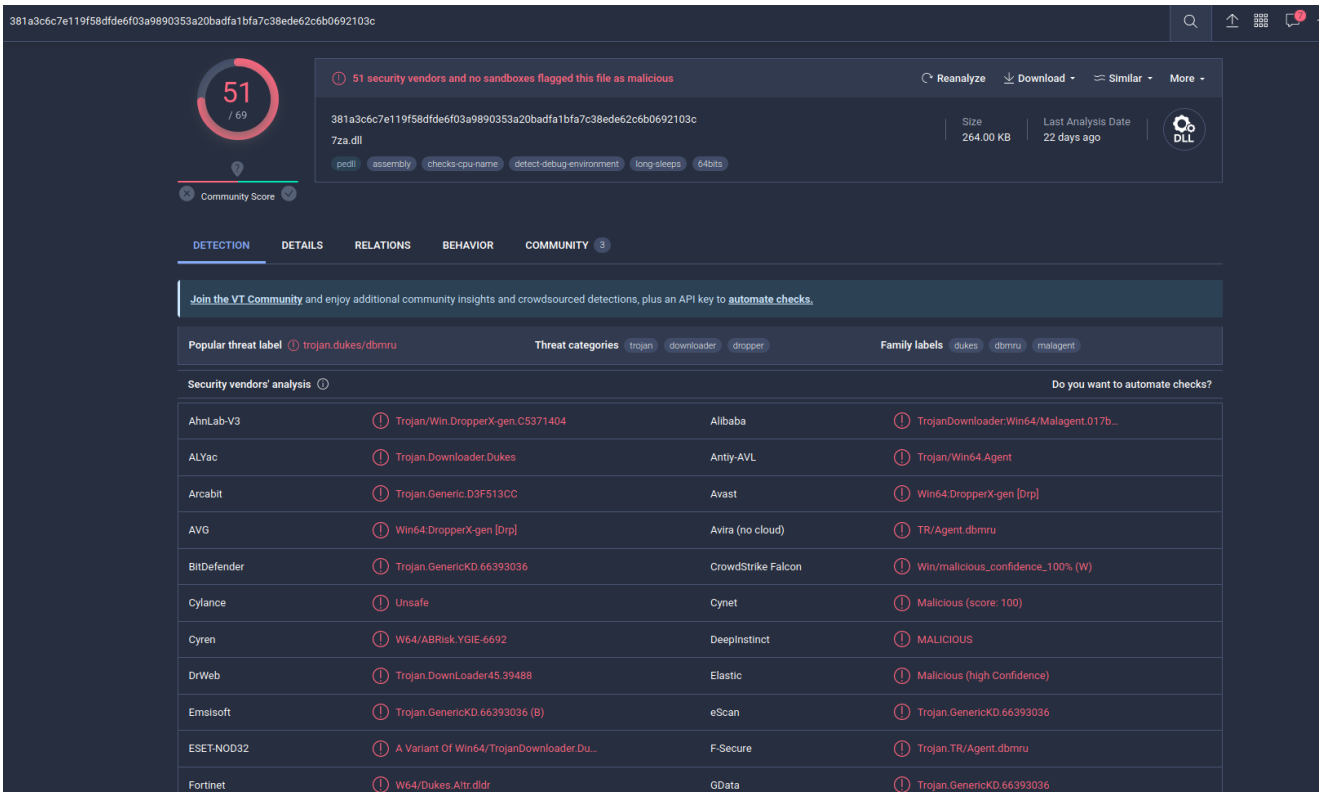
```
0xfdb3b358,//wvsprintfA
0xfdb3b3a6//wvsprintfW
```

sample4.exe - this sample is SNOWYAMBER DLL

File size: `270336 bytes`
MD5 sum: `d0efe94196b4923eb644ec0b53d226cc`
SHA-1 sum: `c938934c0f5304541087313382aee163e0c5239c`
SHA-256 sum: `381a3c6c7e119f58dfde6f03a9890353a20badfa1bfa7c38ede62c6b0692103c`

Checking this sample via VirusTotal:

https://www.virustotal.com/gui/file/381a3c6c7e119f58dfde6f03a9890353a20badfa1bfa7c38ede62c6b0692103c/details



51 of 69 AV engines detect our sample as malicious.

Detected as `Trojan.Downloader.Dukes`.

This sample is well analyzed, the technical details can be viewed here or here.

We will just focus on the fact that this sample uses an interesting string obfuscation technique: using open-source library https://github.com/adamyaxley/Obfuscate

Also used some Conti ransomware tricks like using Murmurhash algorithm.

As we wrote earlier, we believe that the Dukes are a well-resourced, highly dedicated, and well-organized cyberespionage group that has been working for the Russian Federation since at least 2008 to gather intelligence in support of foreign and security policy decisions.

The Dukes target predominantly Western governments and related organizations, including government ministries and agencies, political think tanks, and government subcontractors. Their targets have also included governments of Commonwealth of Independent States members, governments of Asia, Africa, and the Middle East, organizations associated with Chechen extremism, and Russian speakers involved in the illegal trade of controlled substances and narcotics.

`MiniDuke`, `CosmicDuke`, `OnionDuke`, `CozyDuke`, `CloudDuke`, `SeaDuke`, `HammerDuke`, `PinchDuke`, and `GeminiDuke` are examples of the extensive arsenal of malware toolsets utilized by the Dukes. In recent years, the Dukes have evidently conducted large-scale spear-phishing campaigns biannually against hundreds or thousands of recipients affiliated with government institutions and affiliated organizations.

IOCs

Sigma rule

```
title: Remote Thread Creation In Uncommon Target Image
id: a1a144b7-5c9b-4853-a559-2172be8d4a03
related:
    - id: f016c716-754a-467f-a39e-63c06f773987
      type: obsoletes
status: experimental
description: Detects uncommon target processes for remote thread creation
references:
    - https://blog.redbluepurple.io/offensive-research/bypassing-injection-detection
author: Florian Roth (Nextron Systems)
date: 2022/03/16
modified: 2023/05/05
tags:
    - attack.defense_evasion
    - attack.privilege_escalation
    - attack.t1055.003
logsource:
    product: windows
    category: create_remote_thread
detection:
    selection:
        TargetImage|endswith:
            - '\calc.exe'
            - '\calculator.exe'
            - '\explorer.exe'
            - '\mspaint.exe'
            - '\notepad.exe'
            - '\ping.exe'
            - '\sethc.exe'
            - '\spoolsv.exe'
            - '\wordpad.exe'
            - '\write.exe'
    filter_optional_aurora_1:
        StartFunction: 'EtwpNotificationThread'
    filter_optional_aurora_2:
        SourceImage|contains: 'unknown process'
    filter_main_spoolsv:
        SourceImage: 'C:\Windows\System32\csrss.exe'
        TargetImage: 'C:\Windows\System32\spoolsv.exe'
    condition: selection and not 1 of filter_main_* and not 1 of filter_optional_*
falsepositives:
    - Unknown
level: high
```

Conclusion

Running code in the context of another process may allow a threat actor to access the process's memory, system/network resources, and possibly elevated privileges. PE injection is commonly used by malware for persistent infection and evasion of detection.

LoadPE Injection is a technique that involves loading a PE file into the memory of a process. In a typical LoadPE Injection scenario, the following steps occur:

- The malware allocates space in its own process for the PE file.
- The malware reads the PE file from disk into the allocated space.
- The malware resolves import addresses for the PE file.
- The malware creates a remote thread in a target process.
- The malware injects the PE file into the address space of the target process.
- The malware initiates execution of the injected PE file in the target process.

This technique allows malware to avoid many behavioral detection strategies. It allows the malicious PE to be executed without ever being directly loaded or written to the disk, making it more difficult for traditional antivirus software to detect.

As we can see, the technique is not new but is still used in 2023.

We believe that this is either a new modification of *Snowyamber* or a new Conti style malware family, since any Russian related groups use ContiLeaks. ContiLeaks is a turning point in the cybercrime ecosystem, and in this case, we can expect a lot of changes in how cybercriminal organizations operate.

By Cyber Threat Hunters from MSSPLab:

- @cocomelonc
- @wqkasper

References

APT29
SNOWYAMBER Malware Analysis Report
https://github.com/SigmaHQ/sigma
Process Injection
https://github.com/adamyaxley/Obfuscate
Conti ransomware source code investigation - part 1.
Conti ransomware source code investigation - part 2

Thanks for your time happy hacking and good bye!
All drawings and screenshots are MSSPLab's