Fresh PlugX October 2019

SC

On 15 November 2019, I received a VirusTotal notification for a copy of PlugX that had been uploaded (Yara - PlugXBootLDRCode from https://github.com/citizenlab/malware-signatures/blob/master/malware-families/plugx.yara).

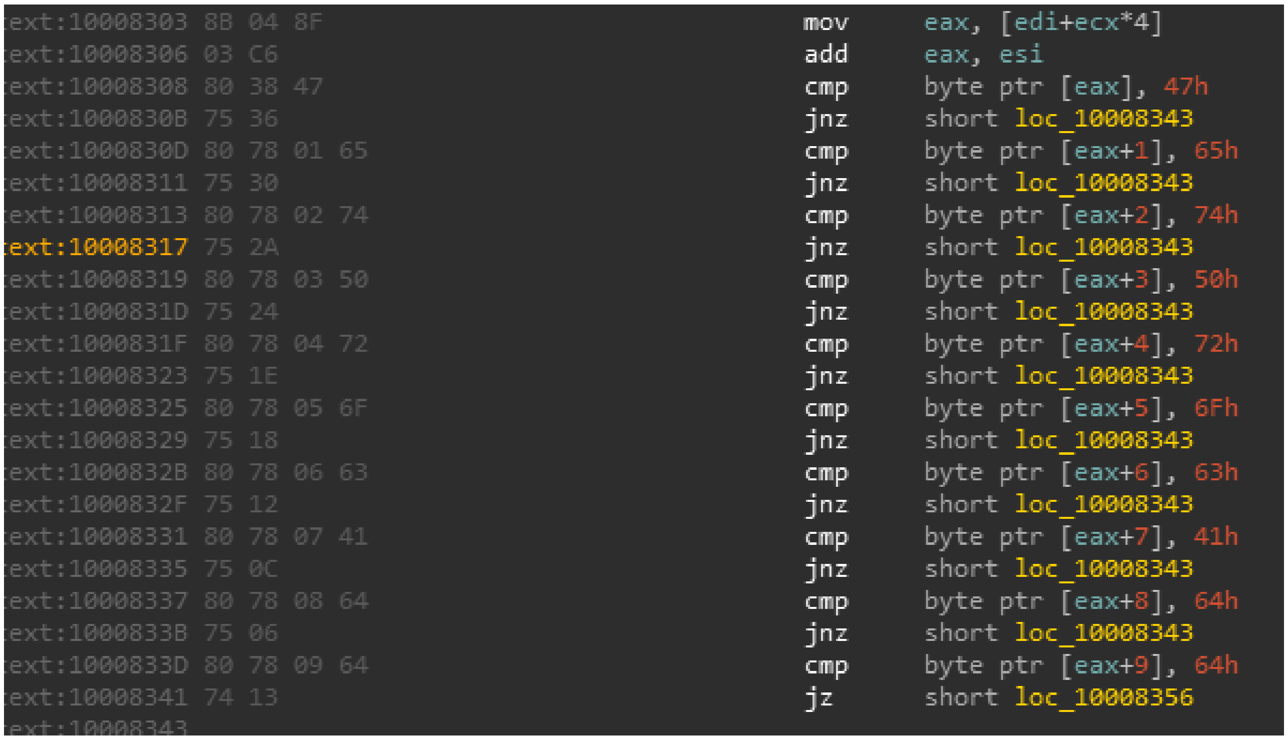
```
MD5          : ce67994a4ee7cf90645e93aec084230d
SHA1          : b42c84f851b8b7d2d2ddfbc9ac94e001204faf45
SHA256        :
6b46e36245b5b9ed13c0fbfae730b49c04aba43b98deb75e388e03695ff5cbd1
Type          : Win32 DLL

First seen   : 2019-11-15 08:04:32 UTC
Last seen    : 2019-11-15 08:04:32 UTC&nbsp

First name   : plugx.dll
```

What stood out from the notification (outside of the file being named plugx.dll) was a compilation time of Fri Oct 4 08:34:45 2019 UTC (a little more then a month before the writing of this post).

Initial Validation

This specific rule matches on operations for assembling a set of API calls - shown below

```
$ yara -s All.yara sample
PlugXBootLDRCode [PlugX,Family]
6b46e36245b5b9ed13c0fbfae730b49c04aba43b98deb75e388e03695ff5cbd1
0x7708:$GetProcAdd: 80 38 47 75 36 80 78 01 65 75 30 80 78 02 74 75 2A 80 78 03 50
0x7786:$L4_LoadLibraryA: C7 85 5C FF FF FF 4C 6F 61 64 C7 85 60 FF FF FF 4C 69 62
0x7859:$L4_ExitThread: C7 85 FC FE FF FF 45 78 69 74 C7 85 00 FF FF FF 54 68 72 65
```



```
ext:10008303 8B 04 8F                mov     eax, [edi+ecx*4]
ext:10008306 03 C6                   add     eax, esi
ext:10008308 80 38 47                cmp     byte ptr [eax], 47h
ext:1000830B 75 36                   jnz     short loc_10008343
ext:1000830D 80 78 01 65             cmp     byte ptr [eax+1], 65h
ext:10008311 75 30                   jnz     short loc_10008343
ext:10008313 80 78 02 74             cmp     byte ptr [eax+2], 74h
ext:10008317 75 2A                   jnz     short loc_10008343
ext:10008319 80 78 03 50             cmp     byte ptr [eax+3], 50h
ext:1000831D 75 24                   jnz     short loc_10008343
ext:1000831F 80 78 04 72             cmp     byte ptr [eax+4], 72h
ext:10008323 75 1E                   jnz     short loc_10008343
ext:10008325 80 78 05 6F             cmp     byte ptr [eax+5], 6Fh
ext:10008329 75 18                   jnz     short loc_10008343
ext:1000832B 80 78 06 63             cmp     byte ptr [eax+6], 63h
ext:1000832F 75 12                   jnz     short loc_10008343
ext:10008331 80 78 07 41             cmp     byte ptr [eax+7], 41h
ext:10008335 75 0C                   jnz     short loc_10008343
ext:10008337 80 78 08 64             cmp     byte ptr [eax+8], 64h
ext:1000833B 75 06                   jnz     short loc_10008343
ext:1000833D 80 78 09 64             cmp     byte ptr [eax+9], 64h
ext:10008341 74 13                   jz      short loc_10008356
ext:10008343
```

Screenshot of match condition in IDA

As a general note, the -s flag in Yara is used for outputting the matched strings and is extremely useful for debugging rules and evaluating why a file matched.

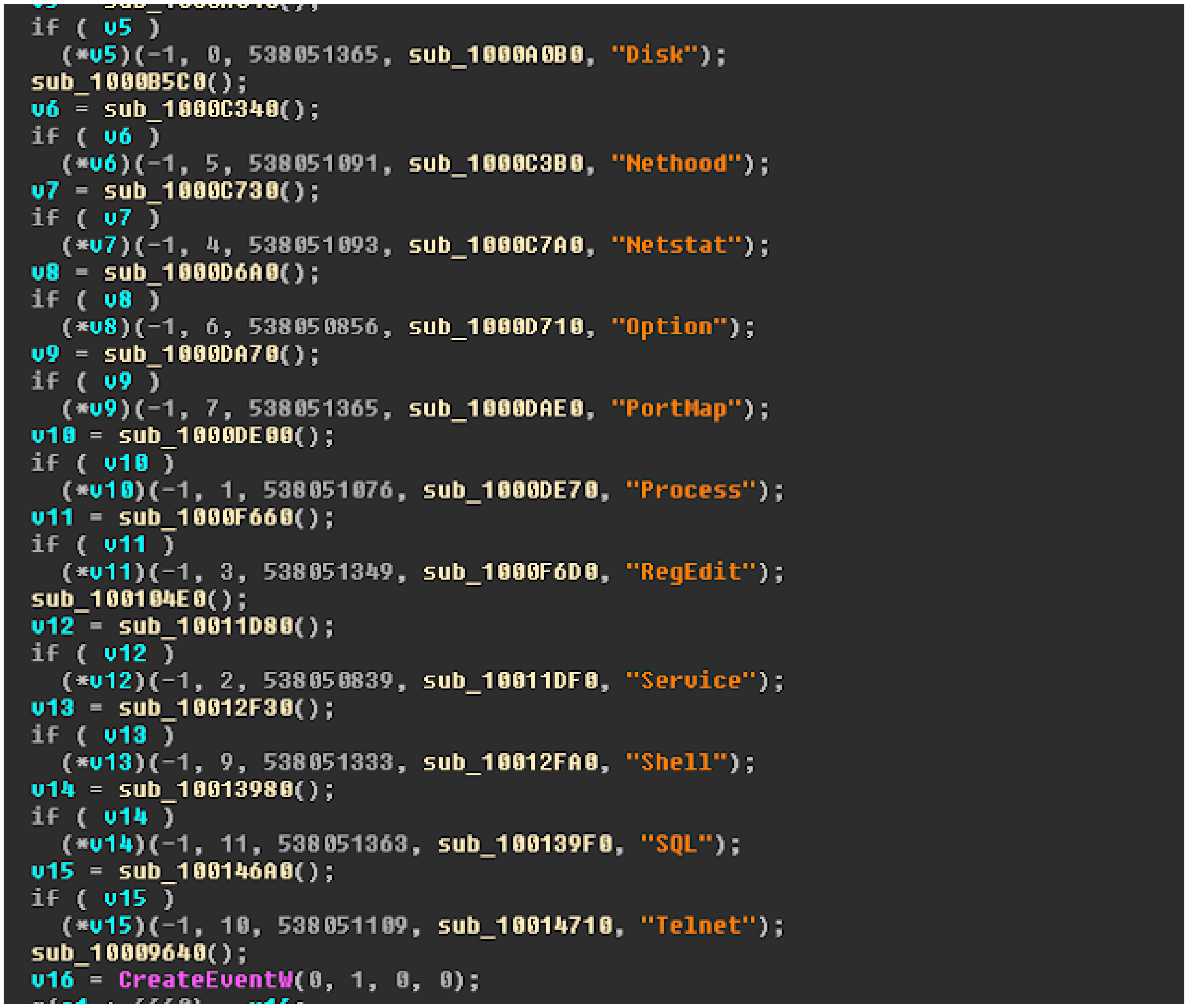
From a quick comparison of the strings, a quick Google search found previous reporting confirming this file was PlugX (ref: http://takahiroharuyama.github.io/blog/2014/03/27/id-slash-idapython-scripts-extracting-plugx-configs/)

DEMO...
THIS IS A DEMO VERSION!!!
\\.\PIPE\RUN_AS_USER(%d)
%WINDIR%\SYSTEM32\SERVICES.EXE
Software\Microsoft\Windows\CurrentVersion\Run
System\CurrentControlSet\Services
debug.hlp
C:\Windows\System32\rundll32.exe "%s" BypassUAC %s
PI[%8.8X]
%s\%d.plg
mytilus3.hlp

%04d-%02d-%02d %02d:%02d:%02d

Overlaps with versions

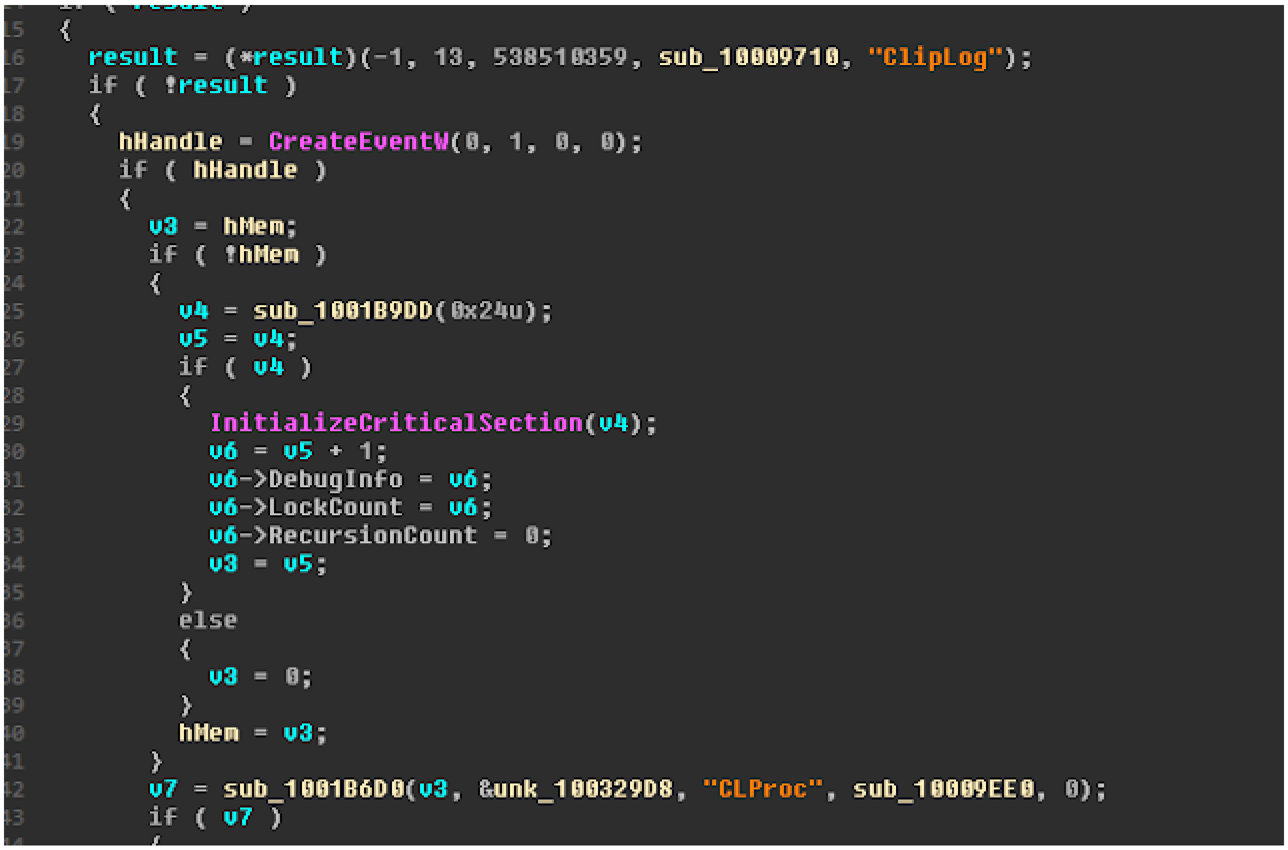
A outstanding point of reference evaluating PlugX is the Sophos report (https://www.sophos.com/en-us/medialibrary/pdfs/technical%20papers/plugx-thenextgeneration.pdf). On Page 7, Gabor Szappanos has a table covering the supported commands. In this copy, sub_10008DE acts as a command handler for evaluating operator commands and can be used to evaluate this copy against that from 2014:



```
  if ( v5 )
    (*v5)(-1, 0, 538051365, sub_1000A0B0, "Disk");
  sub_1000B5C0();
  v6 = sub_1000C340();
  if ( v6 )
    (*v6)(-1, 5, 538051091, sub_1000C3B0, "Nethood");
  v7 = sub_1000C730();
  if ( v7 )
    (*v7)(-1, 4, 538051093, sub_1000C7A0, "Netstat");
  v8 = sub_1000D6A0();
  if ( v8 )
    (*v8)(-1, 6, 538050856, sub_1000D710, "Option");
  v9 = sub_1000DA70();
  if ( v9 )
    (*v9)(-1, 7, 538051365, sub_1000DAE0, "PortMap");
  v10 = sub_1000DE00();
  if ( v10 )
    (*v10)(-1, 1, 538051076, sub_1000DE70, "Process");
  v11 = sub_1000F660();
  if ( v11 )
    (*v11)(-1, 3, 538051349, sub_1000F6D0, "RegEdit");
  sub_100104E0();
  v12 = sub_10011D80();
  if ( v12 )
    (*v12)(-1, 2, 538050839, sub_10011DF0, "Service");
  v13 = sub_10012F30();
  if ( v13 )
    (*v13)(-1, 9, 538051333, sub_10012FA0, "Shell");
  v14 = sub_10013980();
  if ( v14 )
    (*v14)(-1, 11, 538051363, sub_100139F0, "SQL");
  v15 = sub_100146A0();
  if ( v15 )
    (*v15)(-1, 10, 538051109, sub_10014710, "Telnet");
  sub_10009640();
  v16 = CreateEventW(0, 1, 0, 0);
```

In the above screenshot, many of the commands from the 2014 version are present; some additional commands are present, however, handled withing sub-functions of sub_10008DE.

What did appear unique was a set of commands for monitor clipboard activity:



```
{
  result = (*result)(-1, 13, 538510359, sub_10009710, "ClipLog");
  if ( !result )
  {
    hHandle = CreateEventW(0, 1, 0, 0);
    if ( hHandle )
    {
      v3 = hMem;
      if ( !hMem )
      {
        v4 = sub_1001B9DD(0x24u);
        v5 = v4;
        if ( v4 )
        {
          InitializeCriticalSection(v4);
          v6 = v5 + 1;
          v6->DebugInfo = v6;
          v6->LockCount = v6;
          v6->RecursionCount = 0;
          v3 = v5;
        }
        else
        {
          v3 = 0;
        }
        hMem = v3;
      }
      v7 = sub_1001B6D0(v3, &unk_100329D8, "CLProc", sub_10009EE0, 0);
      if ( v7 )
```

An initial Google search did not show any hits for these being previously documented commands in PlugX - suggesting it may be a new feature - however, further analysis is needed to validate this.

Backdooring a HID Reader

Lazarus obfuscation in Feb 2019