ExPetr

A gut feeling of old acquaintances, new tools, and a common battleground

By GReAT on June 30, 2017. 9:39 pm

Much has been written about the recent ExPetr/NotPetya/Nyetya/Petya outbreak – you can read our findings here:Schroedinger's Pet(ya) and ExPetr is a wiper, not ransomware.

As in the case of Wannacry, attribution is very difficult and finding links with previously known malware is challenging. In the case of Wannacry, Google's Neel Mehta was able to identify a code fragment which became the most important clue in the story, and was later confirmed by further evidence, showing Wannacry as a pet project of the Lazarus group.

To date, nobody has been able to find any significant code sharing between ExPetr/Petya and older malware. Given our love for unsolved mysteries, we jumped right on it.

Analyzing the Similarities

At the beginning of the ExPetr outbreak, one of our team members pointed to the fact that the specific list of extensions used by ExPetr is very similar to the one used by BlackEnergy's KillDisk ransomware from 2015 and 2016 (Anton Cherepanov from ESET made the same observation on Twitter).

The BlackEnergy APT is a sophisticated threat actor that is known to have used at least one zero day, coupled with destructive tools, and code geared towards attacking ICS systems. They are widely confirmed as the entity behind the Ukraine power grid attack from 2015 as well as a chain of other destructive attacks that plagued that country over the past years.

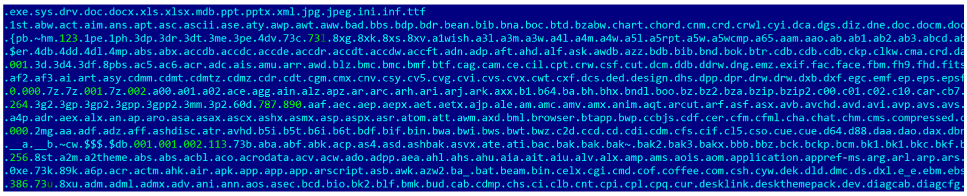
If you are interested in reading more about the BlackEnergy APT, be sure to check our previous blogs on the topic:

- BE2 custom plugins, router abuse and target profiles
- BE2 extraordinary plugins
- BlackEnergy APT Attacks in Ukraine employ spearphishing with Word documents

Going back to the hunt for similarities, here's how the targeted extensions lists looks in ExPetr and a version of a wiper used by the BE APT group in 2015:

ExPetr	2015 BlackEnergy wiper sample
3ds, .7z, .accdb, .ai, .asp, .aspx, .avhd, .back, .bak, .c, .cfg, .conf, .cpp, .cs, .ctl, .dbf, .disk, .djvu, .doc, .docx, .dwg, .eml, .fdb, .gz, .h, .hdd, .kdbx, .mail, .mdb, .msg, .nrg, .ora, .ost, .ova, .ovf, .pdf, .php, .pmf, .ppt, .pptx, .pst, .pvi, .py, .pyc, .rar, .rtf, .sln, .sql, .tar, .vbox, .vbs, .vcb, .vdi, .vfd, .vmc, .vmdk, .vmsd, .vmx, .vsdx, .vsv, .work, .xls	.3ds, .7z, .accdb, .accdc, .ai, .asp, .aspx, .avhd, .back, .bak, .bin, .bkf, .cer, .cfg, .conf, .crl, .crt, .csr, .csv, .dat, .db3, .db4, .dbc, .dbf, .dbx, .djvu, .doc, .docx, .dr, .dwg, .dxf, .edb, .eml, .fdb, .gdb, .git, .gz, .hdd, .ib, .ibz, .io, .jar, .jpeg, .jpg, .jrs, .js, .kdbx, .key, .mail, .max, .mdb, .mdbx, .mdf, .mkv, .mlk, .mp3, .msi, .my, .myd, .nsn, .oda, .ost, .ovf, .p7b, .p7c, .p7r, .pd, .pdf, .pem, .pfx, .php, .pio, .piz, .png, .ppt, .pptx, .ps, .ps1, .pst, .pvi, .pvk, .py, .pyc, .rar, .rb, .rtf, .sdb, .sdf, .sh, .sl3, .spc, .sql, .sqlite, .sqlite3, .tar, .tiff, .vbk, .vbm, .vbox, .vcb, .vdi, .vfd, .vhd, .vhdx, .vmc, .vmdk, .vmem, .vmfx, .vmsd, .vmx, .vmxf, .vsd, .vsdx, .vsv, .wav, .wdb, .xls, .xlsx, .xvd, .zip

Obviously, the lists are similar in composition and formatting, but not identical. Moreover, older versions of the BE destructive module have even longer lists. Here's a snippet of an extensions list from a 2015 BE sample that is even longer:

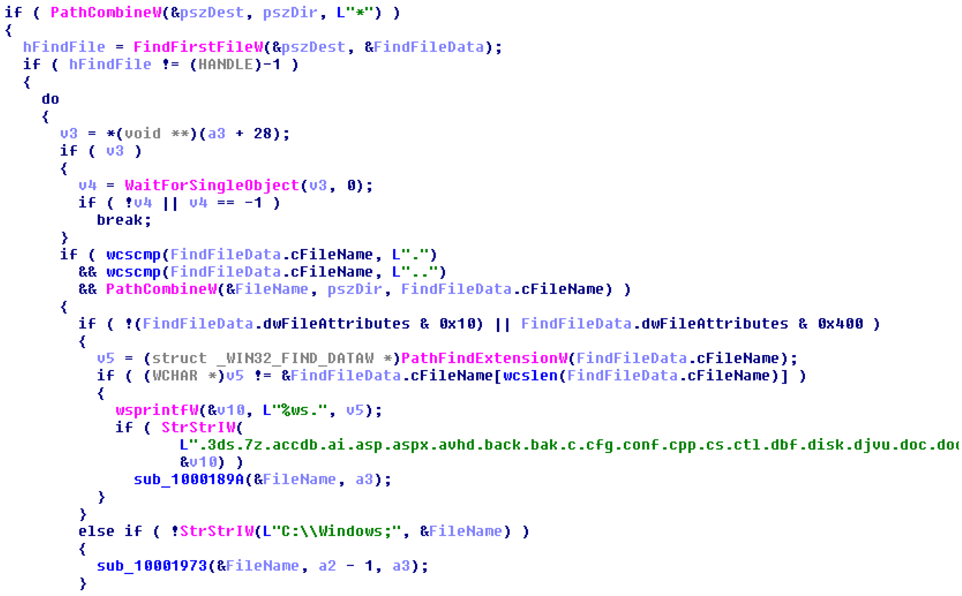


```
.exe.sys.drv.doc.docx.xls.xlsx.mdb.ppt.pptx.xml.jpg.jpeg.ini.inf.ttf
.1st.abw.act.aim.ans.apt.asc.ascii.ase.aty.awp.awt.aww.bad.bbs.bdp.bdr.bean.bib.bna.boc.btd.bzabw.chart.chord.cnm.crd.crwl.cyi.dca.dgs.diz.dne.doc.docm.doc
.{pb.~hm.123.1pe.1ph.3dp.3dr.3dt.3me.3pe.4dv.73c.73l.8xg.8xk.8xs.8xv.a1wish.a3l.a3m.a3w.a4l.a4m.a4w.a5l.a5rpt.a5w.a5wcmp.a65.aam.aao.ab.ab1.ab2.ab3.abcd.ab
.$er.4db.4dd.4dl.4mp.abs.abx.accdb.accdc.accde.accdr.accdt.accdw.accft.adn.adp.aft.ahd.alf.ask.awdb.azz.bdb.bib.bnd.bok.btr.cdb.cdb.cdb.ckp.clkw.cma.crd.da
.001.3d.3d4.3df.8pbs.ac5.ac6.acr.adc.ais.amu.arr.awd.blz.bmc.bmc.bmf.btf.cag.cam.ce.cil.cpt.crw.csf.cut.dcm.ddb.ddrw.dng.emz.exif.fac.face.fbm.fh9.fhd.fits
.af2.af3.ai.art.asy.cdmm.cdmt.cdmtz.cdmz.cdr.cdt.cgm.cmx.cnv.csy.cv5.cvg.cvi.cvs.cvx.cwt.cxf.dcs.ded.design.dhs.dpp.dpr.drw.drw.dxb.dxf.egc.emf.ep.eps.epsf
.0.000.7z.7z.001.7z.002.a00.a01.a02.ace.agg.ain.alz.apz.ar.arc.arh.ari.arj.ark.axx.b1.b64.ba.bh.bhx.bndl.boo.bz.bz2.bza.bzip.bzip2.c00.c01.c02.c10.car.cb7.
.264.3g2.3gp.3gp2.3gpp.3gpp2.3mm.3p2.60d.787.890.aaf.aec.aep.aepx.aet.aetx.ajp.ale.am.amc.amv.amx.anim.aqt.arcut.arf.asf.asx.avb.avchd.avd.avi.avp.avs.avs.
.a4p.adr.aex.alx.an.ap.aro.asa.asax.ascx.ashx.asmx.asp.aspx.asr.atom.att.awm.axd.bml.browser.btapp.bwp.ccbjs.cdf.cer.cfm.cfml.cha.chat.chm.cms.compressed.c
.000.2mg.aa.adf.adz.aff.ashdisc.atr.avhd.b5i.b5t.b6i.b6t.bdf.bif.bin.bwa.bwi.bws.bwt.bwz.c2d.ccd.cd.cdi.cdm.cfs.cif.cl5.cso.cue.cue.d64.d88.daa.dao.dax.dbr
.__a.__b.~cw.$$$.$db.001.001.002.113.73b.aba.abf.abk.acp.as4.asd.ashbak.asvx.ate.ati.bac.bak.bak.bak~.bak2.bak3.bakx.bbb.bbz.bck.bckp.bcm.bk1.bk1.bkc.bkf.b
.256.8st.a2m.a2theme.abs.abs.acbl.aco.acrodata.acv.acw.ado.adpp.aea.ahl.ahs.ahu.aia.ait.aiu.alv.alx.amp.ams.aois.aom.application.appref-ms.arg.arl.arp.ars.
.0xe.73k.89k.a6p.acr.actm.ahk.air.apk.app.app.app.arscript.asb.awk.azw2.ba_.bat.beam.bin.celx.cgi.cmd.cof.coffee.com.csh.cyw.dek.dld.dmc.ds.dxl.e_e.ebm.ebs
.386.73u.8xu.adm.adml.admx.adv.ani.ann.aos.asec.bcd.bio.bk2.blf.bmk.bud.cab.cdmp.chs.ci.clb.cnt.cpi.cpl.cpq.cur.desklink.deskthemepack.dev.diagcab.diagcfg.
```

Nevertheless, the lists were similar in the sense of being stored in the same dot-separated formats. Although this indicated a possible link, we wondered if we could find more similarities, especially in the code of older variants of BlackEnergy and ExPetr.

We continued to chase that hunch during the frenetic early analysis phase and shared this gut feeling of a similarity between ExPetr and BlackEnergy with our friends at Palo Alto Networks. Together, we tried to build a list of features that we could use to make a YARA rule to detect both ExPetr and BlackEnergy wipers.

During the analysis, we focused on the similar extensions list and the code responsible for parsing the file system for encryption or wiping. Here's the code responsible for checking the extensions to target in the current version of ExPetr:



```
if ( PathCombineW(&pszDest, pszDir, L"*") )
{
  hFindFile = FindFirstFileW(&pszDest, &FindFileData);
  if ( hFindFile != (HANDLE)-1 )
  {
    do
    {
      v3 = *(void **)(a3 + 28);
      if ( v3 )
      {
        v4 = WaitForSingleObject(v3, 0);
        if ( !v4 || v4 == -1 )
          break;
      }
      if ( wcscmp(FindFileData.cFileName, L".")
        && wcscmp(FindFileData.cFileName, L"..")
        && PathCombineW(&FileName, pszDir, FindFileData.cFileName) )
      {
        if ( !(FindFileData.dwFileAttributes & 0x10) || FindFileData.dwFileAttributes & 0x400 )
        {
          v5 = (struct _WIN32_FIND_DATAW *)PathFindExtensionW(FindFileData.cFileName);
          if ( (WCHAR *)v5 != &FindFileData.cFileName[wcslen(FindFileData.cFileName)] )
          {
            wsprintfW(&v10, L"%ws.", v5);
            if ( StrStrIW(
                   L".3ds.7z.accdb.ai.asp.aspx.avhd.back.bak.c.cfg.conf.cpp.cs.ctl.dbf.disk.djvu.doc.do
                   &v10) )
              sub_1000189A(&FileName, a3);
          }
        }
        else if ( !StrStrIW(L"C:\\Windows;", &FileName) )
        {
          sub_10001973(&FileName, a2 - 1, a3);
        }
```

This works by going through the target file system in a recursive way, then checking if the extension for each file is included in the dot-separated list. Unfortunately for our theory, the way this is implemented in older BlackEnergy variants is quite different; the code is more generic and the list of extensions to target is initialized at the beginning, and passed down to the recursive disk listing function.

Instead, we took the results of automated code comparisons and paired them down to a signature that perfectly fit the mould of both in the hope of unearthing similarities. What we came up with is a combination of generic code and interesting strings that we put together into a cohesive rule to single out both BlackEnergy KillDisk components and ExPetr samples. The main example of this generic code is the inlined wcscmp function merged by the compiler's optimization, meant to check if the filename is the current folder, which is named ".". Of course, **this code is pretty generic and can appear in other programs** that recursively list files. It's inclusion alongside a similar extension list makes it of particular interest to us –but remains a **low confidence** indicator.

Looking further, we identified some other candidate strings which, although not unique, when combined together allow us to fingerprint the binaries from our case in a more precise way. These include:

- exe /r /f
- ComSpec
- InitiateSystemShutdown

When put together with the wcscmp inlined code that checks on the filename, we get the following YARA rule:

```

rule blackenergy_and_petya_similarities {

strings:
   //shutdown.exe /r /f
   $bytes00 = { 73 00 68 00 75 00 74 00 64 00 6f 00 77 00 6e

   //ComSpec
   $bytes01 = { 43 00 6f 00 6d 00 53 00 70 00 65 00 63 00 }

   //InitiateSystemShutdown
   $bytes02 = { 49 6e 69 74 69 61 74 65 53 79 73 74 65 6d 53

   //68A4430110                     push         0100143A4 ;'
   //FF151CD10010                   call         GetModuleHan
   //3BC7                           cmp          eax,edi
   //7420                           jz          …
   $bytes03 = { 68 ?? ?? ?1 ?0 ff 15 ?? ?? ?? ?0 3b c7 74 ??

   // “/c”
   $bytes04 = { 2f 00 63 00 }

            //wcscmp(…
   $hex_string = { b9 ?? ?? ?1 ?0 8d 44 24 ?c 66 8b 10 66 3b
                   85 d2 74 15 66 8b 50 02 66 3b 51 02 75
                   de 33 c0 eb 05 1b c0 83 d8 ff 85 c0 0f
                   44 24 ?c 66 8b 10 66 3b 11 75 1e 66 85
                   75 0f 83 c0 04 83 c1 04 66 85 d2 75 de
                   0f 84 ?? 0? 00 00 }

condition:

   ((uint16(0) == 0x5A4D)) and (filesize < 5000000) and
   (all of them)
}
```

When run on our extensive (read: very big) malware collection, **the YARA rule above fires on BlackEnergy and ExPetr samples only**. Unsurprisingly, when used alone, each string can generate false positives or catch other unrelated malware. **However, when combined together in this fashion, they become very precise**. The technique of grouping generic or popular strings together into unique combinations is one of the most effective methods for writing powerful Yara rules.

Of course, this should not be considered a sign of a **definitive** link, but it does point to certain code design similarities between these malware families.

This low confidence but persistent hunch is what motivates us to ask **other researchers around the world to join us in investigating these similarities and attempt to discover more facts** about the origin of ExPetr/Petya. Looking back at other high profile cases, such as the Bangladesh Bank Heist or Wannacry, there were few facts linking them to the Lazarus group. In time, more evidence appeared and allowed us, and others, to link them together with high confidence. Further research can be crucial to connecting the dots, or, disproving these theories.

We'd like to think of this ongoing research as an opportunity for an open invitation to the larger security community to help nail down (or disprove) the link between BlackEnergy and ExPetr/Petya. Our colleagues at ESET have published their own excellent analysis suggesting a possible link between ExPetr/Petya and TeleBots (BlackEnergy). Be sure to check out their analysis. And as mentioned before, a special thanks to our friends at Palo Alto for their contributions on clustering BlackEnergy samples.

Hashes

ExPetr:

027cc450ef5f8c5f653329641ec1fed91f694e0d229928963b30f6b0d7d3a745

BE:

11b7b8a7965b52ebb213b023b6772dd2c76c66893fc96a18a9a33c8cf125af80

5d2b1abc7c35de73375dd54a4ec5f0b060ca80a1831dac46ad411b4fe4eac4c6

F52869474834be5a6b5df7f8f0c46cbc7e9b22fa5cb30bee0f363ec6eb056b95

368d5c536832b843c6de2513baf7b11bcafea1647c65df7b6f2648840fa50f75

A6a167e214acd34b4084237ba7f6476d2e999849281aa5b1b3f92138c7d91c7a

Edbc90c217eebabb7a9b618163716f430098202e904ddc16ce9db994c6509310

F9f3374d89baf1878854f1700c8d5a2e5cf40de36071d97c6b9ff6b55d837fca

APT BLACKENERGY DATA ENCRYPTION PETYA WIPER

Share post on:

Related Posts

BlackOasis APT and new targeted attacks leveraging zero-day exploit

The Festive Complexities of SIGINT-Capable Threat Actors

Introducing WhiteBear