Rerverse Engineer: Lab 5 Report

Jesus Lopez, Alejandro Rodriguez, Eduardo Menendez
Dr. Saidur Rahman

November 3, 2024

Contents

Chapter 1

Introduction

In this lab we are going to be exploring the tools of **Volatility** and **Dshell**. These are great cybersecurity tools that are used in the real world and also in some CTF's. For the purpose or doing this lab, we used Kali linux, the system specification is the following:

The instructions on how to install each one of the tools are going to be on each one of their sections, as well a guide on how to use them.

Chapter 2

Volatility

2.1 Introduction

Volatility3 is a very powerful memory forensics tool, that is used to extract memory images of Windows, MacOS, and Linux systems. of course, it is reasonable to mention that there is a huge community writing third-party plugins for volatility3.

2.2 Installation

To install volatility we have to first clone the main repository from their github. This is the command to use:

```
1 git clone https://github.com/volatilityfoundation/volatility3
    .git
```

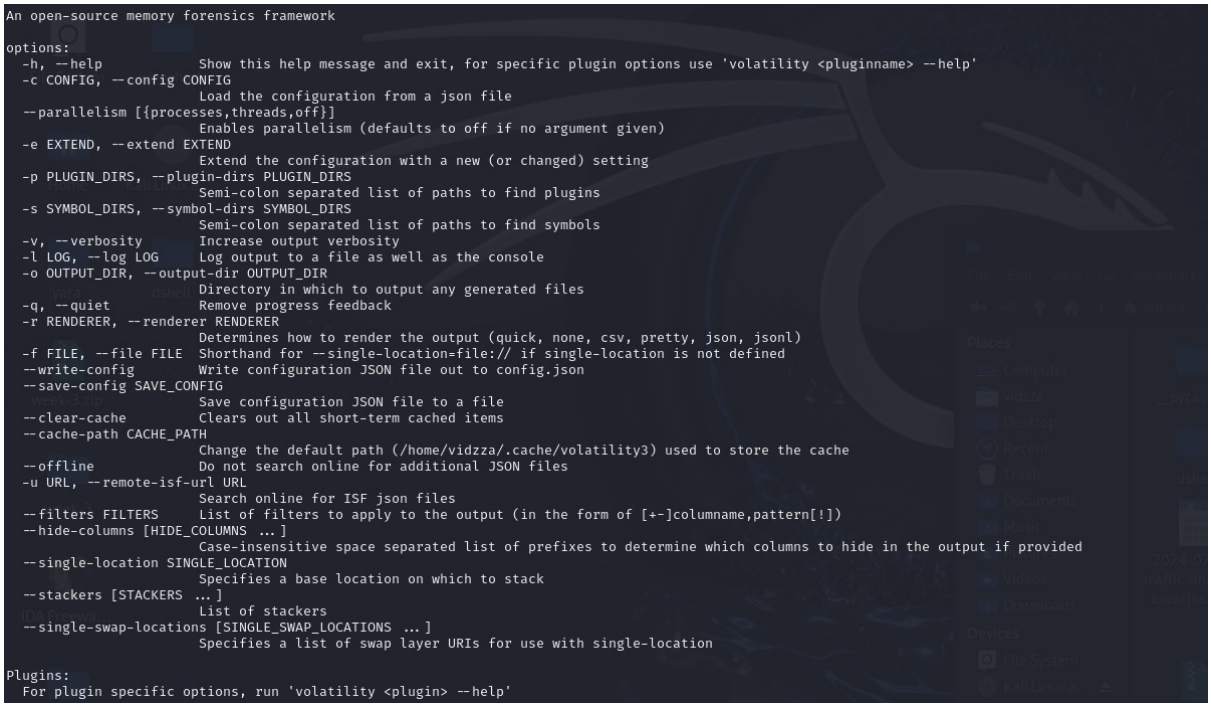
From here there are two options:

1. You can extract it and then let it distribute everything in your system, so that you can use the tool anywhere you like.
2. Call vol.py from the folder that was extracted everytime you like to use the tool.

The main difference is that the first one is quite difficult to remove all the files and clean it, where the second one you only need to erase the folder to erase the tool from your system. For installation purposes, if having trouble you can refer to this page INSTALLATION PAGE

2.2.1 Usage

Here are the options of volatility3:



```
An open-source memory forensics framework

options:
  -h, --help            Show this help message and exit, for specific plugin options use 'volatility <pluginname> --help'
  -c CONFIG, --config CONFIG
                        Load the configuration from a json file
  --parallelism [{processes,threads,off}]
                        Enables parallelism (defaults to off if no argument given)
  -e EXTEND, --extend EXTEND
                        Extend the configuration with a new (or changed) setting
  -p PLUGIN_DIRS, --plugin-dirs PLUGIN_DIRS
                        Semi-colon separated list of paths to find plugins
  -s SYMBOL_DIRS, --symbol-dirs SYMBOL_DIRS
                        Semi-colon separated list of paths to find symbols
  -v, --verbosity       Increase output verbosity
  -l LOG, --log LOG     Log output to a file as well as the console
  -o OUTPUT_DIR, --output-dir OUTPUT_DIR
                        Directory in which to output any generated files
  -q, --quiet           Remove progress feedback
  -r RENDERER, --renderer RENDERER
                        Determines how to render the output (quick, none, csv, pretty, json, jsonl)
  -f FILE, --file FILE  Shorthand for --single-location=file:// if single-location is not defined
  --write-config        Write configuration JSON file out to config.json
  --save-config SAVE_CONFIG
                        Save configuration JSON file to a file
  --clear-cache         Clears out all short-term cached items
  --cache-path CACHE_PATH
                        Change the default path (/home/vidzza/.cache/volatility3) used to store the cache
  --offline             Do not search online for additional JSON files
  -u URL, --remote-isf-url URL
                        Search online for ISF json files
  --filters FILTERS     List of filters to apply to the output (in the form of [+-]columname,pattern[!])
  --hide-columns [HIDE_COLUMNS ...]
                        Case-insensitive space separated list of prefixes to determine which columns to hide in the output if provided
  --single-location SINGLE_LOCATION
                        Specifies a base location on which to stack
  --stackers [STACKERS ...]
                        List of stackers
  --single-swap-locations [SINGLE_SWAP_LOCATIONS ...]
                        Specifies a list of swap layer URIs for use with single-location

Plugins:
  For plugin specific options, run 'volatility <plugin> --help'
```

If desired, you can look at an introduction of memory forensics with volatility [LINK]

For these exercises we are going to be using images that where provided by the instructor.

2.3 Exercise 1: Identify Profile and List Processes

2.3.1 Objective

For task, Use the imageinfo and pslist plugins to identify the OS profile and list all active processes.

2.3.2 Taks

1. Run the following command to determine the correct OS profile:

```
volatility -f example.mem imageinfo
```

2. Use the recommended profile to list all running processes:

```
volatility -f example.mem --profile=Win7SP1x64 pslist
```

3. Identify the top 3 processes consuming the most resources
4. Research any unfamiliar process names and note their purpose.
5. Record the PID, PPID, and creation time of any suspicious processes.

2.3.3 Results

first we are going to run:

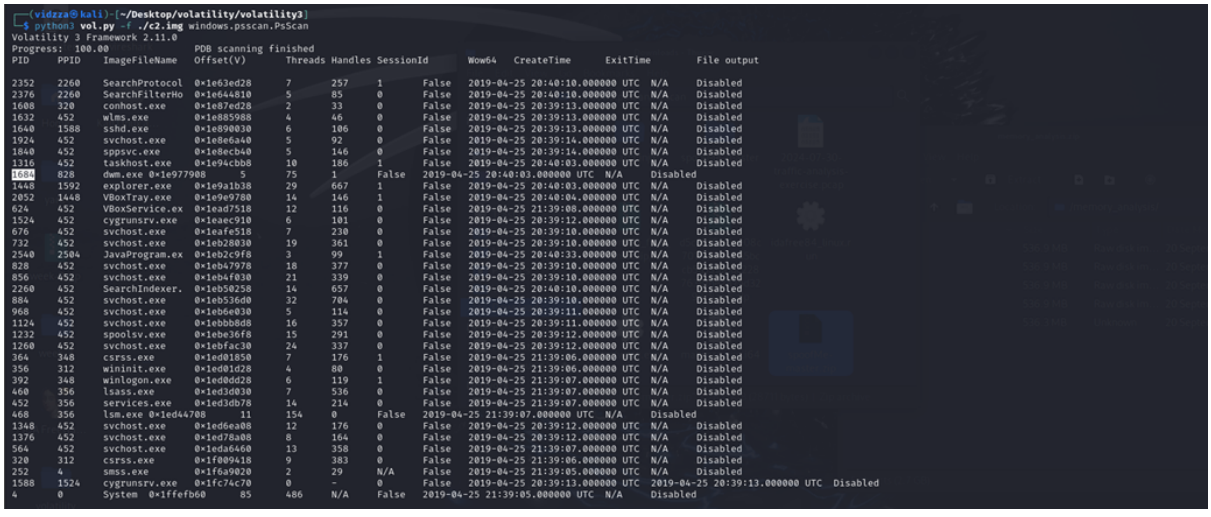
```
sudo vol -f c2.img windows.info.Info
```

This is the result:

to check for processes we are going to use the following command:

```
sudo vol -f c2.img windows.psscan.PsScan
```

from there we are going to get a table with the PID, the PPID, the ImagefileName and more things:



Now, to find the processes that consumes the most, here is a ranking based on handle counts:

1. svchost.exe (PID 884) - 704 Handles
2. explorer.exe (PID 1448) - 667 Handles
3. SearchIndexer.exe (PID 2260) - 657 Handles

For the last section of this task, some weird processes can be ***sshd.exe*** and ***JavaProgram.exe***, the sshd is a process from OpenSSH, the problem might begin if the system was not intended to run SSH services, this might be an indicative that the computer is compromised and has remote access. from the otherside, JavaProgra.exe lacks of descroption on what is doing, this might be a malware running disguissed as a java appliacation.

2.4 Exercise 2:Detect Hidden Processes with psscan and psxview

2.4.1 Objective

for the exercise 2, Use psscan and psxview plugins to detect hidden or terminated processes.

2.4.2 Taks

1. Run psscan to detect hidden and terminated processes:
```
sudo vol -f c2.img windows.psscan.PsScan
```
2. Compare the output with pslist. Identify processes in psscan but not in pslist.
3. Use psxview to find discrepancies between visibility views:
```
sudo vol -f c2.img windows.psxview
```
4. List any processes found by psscan but not in pslist
5. Identify discrepancies in psxview output and explain potential reasons.
6. Record the PID and offset of any suspicious processes.

2.4.3 Results

Something to mention is that on the first exercise we used psscan, it kind of gives us the same results. in this image, we did not noticed any hidden processes, pslist wont list compared to psscan. to run psxview we are going to be using the next command:

```
sudo vol -f c2.img windows.psxview
```

The result of running this command is the following:

```
┌──(vidzza㉿kali)-[~/Desktop/volatility/volatility3]
└─$ python3 vol.py -f ./c2.img psxview
Volatility 3 Framework 2.11.0
Progress:  100.00               PDB scanning finished
Offset(Virtual) Name    PID     pslist  psscan  thrdscan        csrss   Exit Time

0×84fe9780      VBoxTray.exe    2052    True    False   True    True
0×1e9e9780      VBoxTray.exe    2052    False   True    False   False
0×1ed01d28      wininit.exe     356     False   True    False   False
0×84b44708      lsm.exe 468     True    False   True    True
0×84b6ea08      svchost.exe     1348    True    False   True    True
0×84b78a08      svchost.exe     1376    True    False   True    True
0×84e85988      wlms.exe        1632    True    False   True    True
0×84f77908      dwm.exe 1684    True    False   True    True
0×1e885988      wlms.exe        1632    False   True    False   False
0×1e977908      dwm.exe 1684    False   True    False   False
0×1ed44708      lsm.exe 468     False   True    False   False
0×84de36f8      spoolsv.exe     1232    True    False   True    True
0×84cec910      cygrunsrv.exe   1524    True    False   True    True
0×85044810      SearchFilterHo  2376    True    False   True    True
0×1e644810      SearchFilterHo  2376    False   True    False   False
0×1eaec910      cygrunsrv.exe   1524    False   True    False   False
0×1ebe36f8      spoolsv.exe     1232    False   True    False   False
0×84cd7518      VBoxService.ex  624     True    False   True    True
0×84cfe518      svchost.exe     676     True    False   True    True
0×84609418      csrss.exe       320     True    False   True    False
0×1ead7518      VBoxService.ex  624     False   True    False   False
0×1eafe518      svchost.exe     676     False   True    False   False
0×1ed0dd28      winlogon.exe    392     False   True    False   False
0×1f009418      csrss.exe       320     False   True    False   False
0×840a9020      smss.exe        252     True    False   True    False
0×1f6a9020      smss.exe        252     False   True    False   False
0×1eb47978      svchost.exe     828     False   True    False   False
0×84b01d28      wininit.exe     356     True    False   True    True
0×84b0dd28      winlogon.exe    392     True    False   True    True
0×84e7ed28      conhost.exe     1608    True    False   True    True
0×8503ed28      SearchProtocol  2352    True    False   True    True
0×84d2c9f8      JavaProgram.ex  2540    True    False   True    True
0×1e63ed28      SearchProtocol  2352    False   True    False   False
0×1e87ed28      conhost.exe     1608    False   True    False   False
0×1eb2c9f8      JavaProgram.ex  2540    False   True    False   False
0×84b3d030      lsass.exe       460     True    False   True    True
0×84d28030      svchost.exe     732     True    False   True    True
0×84d4f030      svchost.exe     856     True    False   True    True
0×84d6e030      svchost.exe     968     True    False   True    True
0×84b3db78      services.exe    452     True    False   True    True
0×84dfac30      svchost.exe     1260    True    False   True    True
```

Something to mention in the discrepancies between the commands psscan and psxview, is that on psxview it gives us the virtual offset of the process. where on the other command gives us PPID along with the PID. in addition to this on psscan we can get the threads and the handles where on psxview we do not. also, We as a team think that on the columns we can see which of the process can be able to be seen using pslist and some of them can be seen using psscan.

For the PID and the offset of suspicious processes, we are going to be using the ones from the first task. so for sshd.exe is the following:

```
┌──(sagar㉿kali)-[~/Desktop/memory_analysis]
└─$ sudo vol -f c2.img windows.psxview | grep sshd
[sudo] password for sagar:
0×84e90030 100.0sshd.exe        1640    True    False   True    True
0×1e890030      sshd.exe        1640    False   True    False   False
```

and JavaProgram.exe:

```
┌──(sagar㉿kali)-[~/Desktop/memory_analysis]
└─$ sudo vol -f c2.img windows.psxview | grep Java
0×84d2c9f8 100.0JavaProgram.ex  2540    True    False   True    True
0×1eb2c9f8      JavaProgram.ex  2540    False   True    False   False
```

2.5 Exercise 3

2.5.1 Objective

Use the netscan plugin to analyze network activity and extract DLLs and handles.

2.5.2 Taks

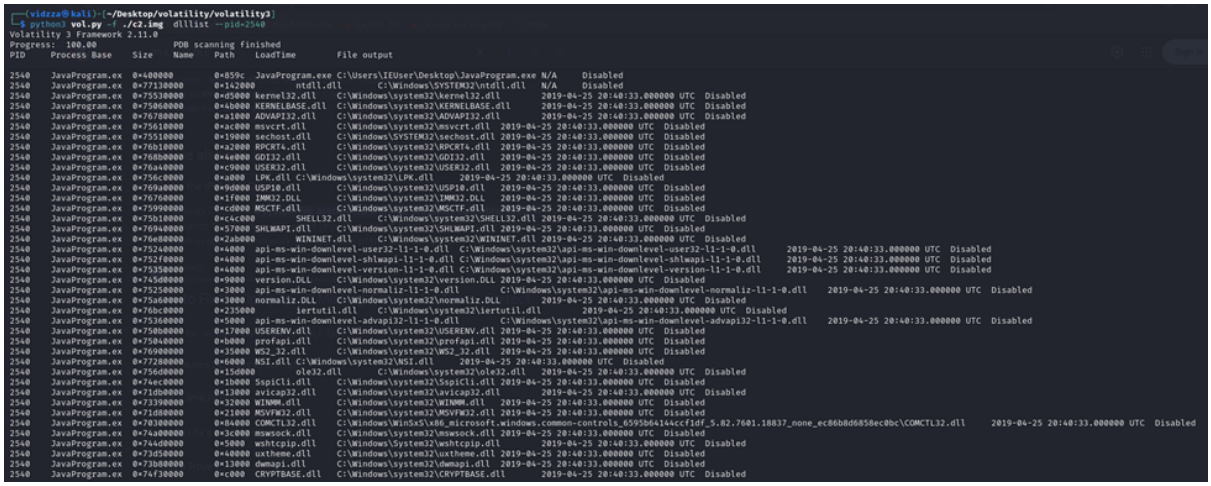
1. Identify suspicious external IPs and their associated processes.
2. List the DLLs and handles associated with the suspicious process.
3. Write a summary explaining any suspicious activity observed.

2.5.3 Results

first we are going to do the identification of suspicious external IPs, it raised suspicion regarding its network activity. Now, the second part is to list the Dlls and handles associated with the suscpicion process.

In this case, the process is ***JavaProgram.exe*** and these are the dlls:

- $C:\backslash Windows\backslash system32\backslash kerne132.dll$
- $C:\backslash Windows\backslash system32\backslash ADVAP132.dll$
- $C:\backslash Windows\backslash system32\backslash USER32.dll$
- $C:\backslash Windows\backslash system32\backslash msvcrt.dll$
- $C:\backslash Windows\backslash system32\backslash sechost.dll$
- $C:\backslash Windows\backslash system32\backslash RPCRT4.dll$



dlls

Now, let us explain what was happening, it seems like the combination of the execution of JavaProgram.exe and the loaded Dlls implies a likelihood of malicious program. of course this is supported by the evidence that the program was trying/attempting to establish persistent connections to unidentified external IPs.

Chapter 3

Dshell

3.1 Introduction

Dshell is an extensible network forensic analysis framework. This tool enables rapid development of plugins to support the dissection of network packet captures.

It is reasonable to mention that devcom list some key features:

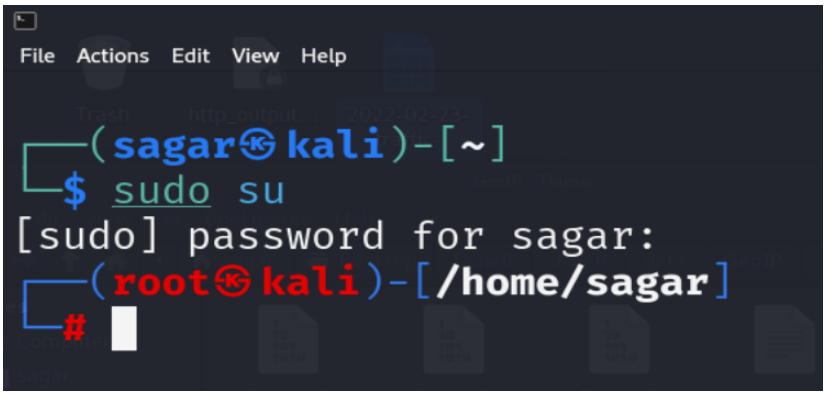
- Deep packet analysis using specialized plugins
- Robust stream reassembly
- IPv4 and IPv6 support
- Multiple user-selectable output formats and the ability to create custom output handlers
- Chainable plugins
- Parallel processing option to divide the handling of data source into separate Python processes
- Enables development of external plugin packs to share and install new externally developed plugins without overlapping the core Dshell plugin directories

3.2 Installation

The installation of this tool is quite different from the one on the presentations, first we have to start by cloning the main repository DSHELL REPO. After that we have to go to the terminal and type

```
1 sudo su
```

This is to put the terminal on root mode, of course it is reasonable to mention that we have to input the credentials of the user to get root privileges. the result should look something like this:



The next step is to install the Dshell with the following command:

```
python3 -m pip install Dshell
```

This should be enough for the instalation. In the end, to start the Dshell it is sufficient to mention that we have to type ***dhsell*** on the terminal.

3.2.1 Usage

For the usage, the commands has change a little bit, for instance, the PDf from the instructor recommends to use the following command:

```
    dshell http -r example.pcap --output http output.json
```

For further use of this tool, here is the reference manual that is provided by devcom on how to use Dshell. MANUAL

As an extra, if desirable, the user can create its own set of features or contribute to the project, here is the Dshell Developer Guide

3.3 Disclaimer

For this tool, we reused one of the PCAPs that we had in the beginning of the semester, of course it is reasonable to mention that we already know the output, but it is interesting how it behaves with another tool that also accepts PCAPS

3.4 Exercise 1: Analyze HTTP Traffic

3.4.1 Objective

The objective of exercise 1 is to use the http decoder in Dshell to identify and extract HTTP communications from a PCAP file.

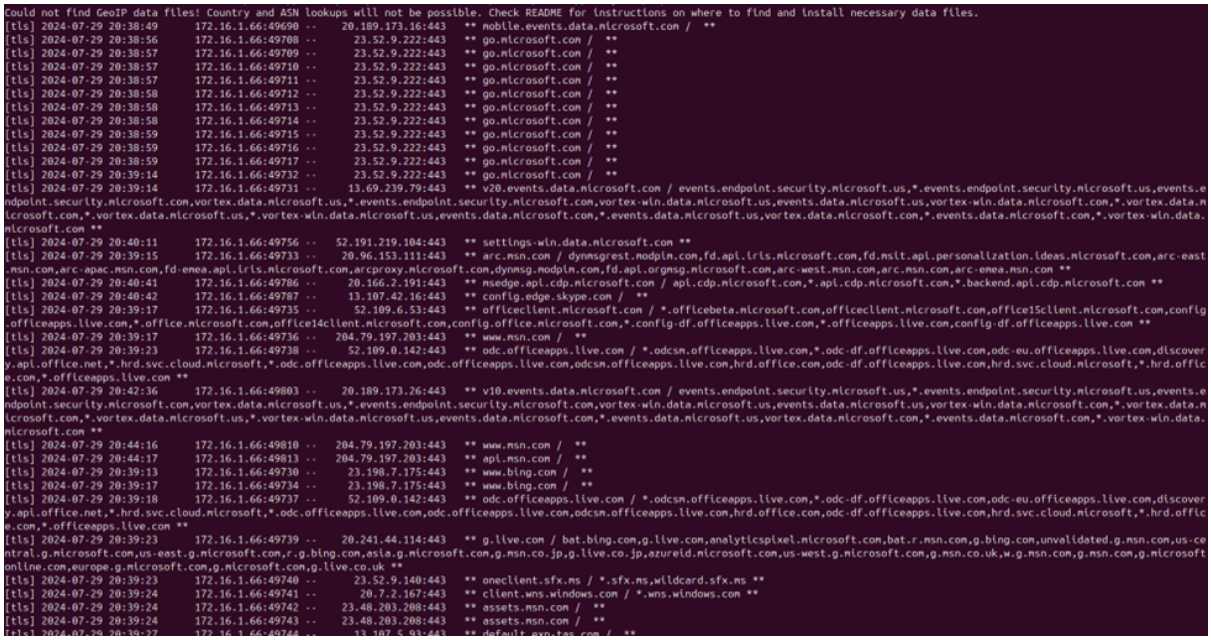
3.4.2 Task

These are the tasks to perform for the first exercise:

1. Use the HTTP decoder to analyze a sample PCAP file.
2. Open the generated Json file and review the HTTP requests and responses.
3. Look for any suspicious URLs, unusual user agents, or large payloads.
4. identify the most frequently requested URL from PCAP.
5. Note the user agents used in the HTTP requests and look for anomalies.
6. Summarize any suspicious findings, such as unexpected data uploads or malicious URLs.

3.4.3 Results

Well, First we are going to identify the most frequently requested URL from the PCAP.



Dshell output after running the analysis

As we can see the most request URL in the PCAP was **go.microsoft.com**, from this URL we can assume that the person who was using the network was trying to access their email, after this the user also goes to the settings of it and then logouts this URL is accesses 10 times over the whole PCAP.

Now, we have to note the user agents used in the HTTP requests and look for anomalies. Something interesting that got out attention is that after going through the analysis, we found that a repository from github was cloned, installed and everything that was on that repo was installed. We suspect that of course, something was installed after the user accesses a link that was infected, afterwards a malware was installed to the user's system.

In the end, the most frequently requested URL in the PCAP we had was **go.microsoft.com**, indicating that the user was likely trying to access their email and subsequently checked settings before logging out. In total, the URL was accessed ten times through the capture.

It is worth mentioning, that the installation we had from Dshell was behaving weirdly, for example, we had a problem with the HTTP plugin, this prevented us from collecting that information, and we tried on two more virtual machines and the problem might originate in the installation process or the distribution, because it was tried on Kali linux, ubuntu, and fedora.

3.5 Exercise 2: Detect DNS Tunneling

3.5.1 Objective

The objective of exercise 2 is to use the dns decoder to identify DNS tunneling or exfiltration attempts.

3.5.2 Task

These are the tasks to perform for the second exercise:

1. Use the dns decoder to analyze PCAP file for DNS queries
2. we have to open the csv file and examine the queried domaind and reponse sizes.
3. we have to look for patterns suggesting DNS tunneling
4. we have to identify domains with unnusually long or complex names.

5. Check for multiple queries to the same domain within a short time frame.

6. Summarize findings and explain whether DNS tunneling is suspected.

3.5.3 Results

For this second exercise, we are required to use the DNs decoder to analyze the PCAP file, Well if we want to do this we have to use this command:

```
decode -p dns 2022-02-23-traffic-analysis-exercise.pcap
```

The output should be the following:

from here we got a CSV file that is called **output.csv**, This will be helpful to analyze patterns,etc.

Well, we can notice that requests, frequently comes from the ip **172.16.0.131 and 172.16.0.170** and these are directed to the DNS server **172.16.0.52**. From this we can infere that 172.16.0.52 serves as a DNS server within the internal network, this helps within **sunnystation.com** domain.

Something else we can notice is that some responses are marked as refused, this could indicate restricted permissions or DNS filtering policies are in place.