

A complex network graph with red lines and white nodes on a dark background, representing a malicious network data analysis.

Malicious Network Data Analysis Using Open-Source Tools

Olivier Bilodeau
Cybersecurity Research Director
GoSecure
@obilodeau

#GOSEC

While waiting, make sure you have a GitHub account to access the workshop environment

GitHub create account (if you don't already have one): <https://github.com/join>

Workshop environment: <https://data-workshop.gosec.co>

Lab Outline

Express edition!!

- **Section 1 – Contextual Information (~15 minutes)**
- **Section 2 – Pcap analysis and data extraction (~40 minutes)**
 - Lab 0 – Introduction to Jupyter Notebook
 - Lab 1 – Explore with Wireshark and extract with Tshark (Wireshark's command-line interface)
 - Lab 2 – Scale Pcap data extraction with GNU parallel

15-minute break

- **Section 3 – Data manipulation and graphs (~50 minutes)**
 - Lab 3 – Manipulate dataframes with network traffic with Pandas
 - Lab 4 – Graph data using hvplot

Who am I?

Olivier Bilodeau

- Cybersecurity Research Director at GoSecure Inc.
- Hacker Jeopardy host for the NorthSec Conference and CTF
- International public speaker at events like RSAC, BlackHat USA, SecTor, HackFest, etc.



The Workshop's Principles

- Be respectful: do not hack the environment
- Ask questions
- Collaborate
- Have fun!

Disclaimer and Copyright

This presentation is part of a workshop conducted by GoSecure Inc.

Permission to reprint/republish this material or to reuse any part of this work (outside of your own organizations) must be obtained from the authors.

This material including associated Pcaps and traffic logs are **TLP:Amber**. If you are not familiar with the Traffic Light Protocol you can read about it here: <https://www.us-cert.gov/tlp>. Code samples are under the simplified BSD license and may be re-used without permission.

For further information or clarification, please contact obilodeau@gosecure.net.

Contextual Information

For years, we have investigated

The Linux/Moose IoT Botnet



**Linux/Moose
Botnet**

That conducted social media fraud : *mainly likes and follows online!*



Linux/Moose Botnet



Stealthy



Constantly
adapting



No direct
victims



Hiding in
plain sight



Large potential
profitability

The Linux/Moose Botnet in 2016



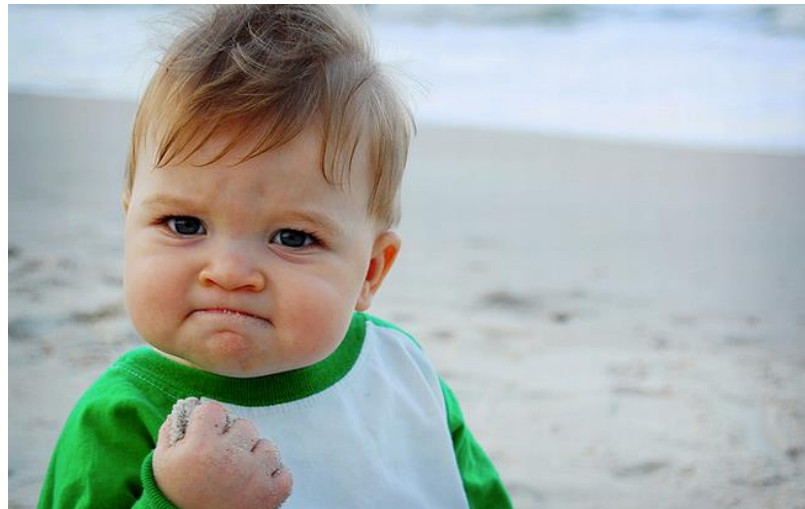
Other parts of the SMM ecosystem studied in 2019

Today

We take a step back...

- Start by explaining the honeypot infrastructure and the data collected on the Linux/Moose botnet
- Analyze some of the traffic together

All this with open-source tools !



Linux/Moose in a Nutshell

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
2:8FB0h:	72	6F	72	00	70	61	73	73	77	6F	72	64	20	69	73	20	ror.password is
2:8FC0h:	77	72	6F	6E	67	00	00	00	70	61	73	73	77	6F	72	64	wrong...password
2:8FD0h:	3A	00	00	00	75	74	68	65	6E	74	69	63	61	74	69	6F	:...uthenticatio
2:8FE0h:	6E	20	66	61	69	6C	65	64	00	00	00	00	73	68	0D	0A	n failed....sh..
2:8FF0h:	00	00	00	00	70	73	0D	0A	65	63	68	6F	20	2D	6E	20ps..echo -n
2:9000h:	2D	65	20	22	48	33	6C	4C	30	57	6F	52	6C	44	22	0D	-e "H3lL0WoRlD".
2:9010h:	0A	63	68	6D	6F	64	0D	0A	00	00	00	00	48	33	6C	4C	.chmod.....H3lL
2:9020h:	30	57	6F	52	6C	44	00	00	65	6C	61	6E	32	00	00	00	0WoRlD..elan2...
2:9030h:	65	6C	61	6E	33	00	00	00	63	68	6D	6F	64	3A	20	6E	elan3...chmod: n
2:9040h:	6F	74	20	66	6F	75	6E	64	00	00	00	00	63	61	74	20	ot found....cat
2:9050h:	2F	70	72	6F	63	2F	63	70	75	69	6E	66	6F	0D	0A	00	/proc/cpuinfo...
2:9060h:	47	45	54	20	2F	78	78	2F	72	6E	64	65	2E	70	68	70	GET /xx/rnde.php
2:9070h:	3F	70	3D	25	64	26	66	3D	25	64	26	6D	3D	25	64	20	?p=%d&f=%d&m=%d
2:9080h:	48	54	54	50	2F	31	2E	31	0D	0A	48	6F	73	74	3A	20	HTTP/1.1..Host:
2:9090h:	77	77	77	2E	67	65	74	63	6F	6F	6C	2E	63	6F	6D	0D	www.getcool.com.
2:90A0h:	0A	43	6F	6E	6E	65	63	74	69	6F	6E	3A	20	4B	65	65	.Connection: Kee
2:90B0h:	70	2D	41	6C	69	76	65	0D	0A	0D	0A	00	6C	6F	00	00	p-Alive.....lo..
2:90C0h:	31	32	37	2E	30	2E	30	2E	31	00	00	00	2F	70	72	6F	127.0.0.1.../pro
2:90D0h:	63	00	00	00	2F	70	72	6F	63	2F	25	73	2F	63	6D	64	c.../proc/%s/cmd
2:90E0h:	6C	69	6E	65	00	00	00	00	6B	69	6C	6C	20	25	73	00	line....kill %s.
2:90F0h:	2F	65	74	63	2F	69	6E	69	74	2E	64	2F	72	63	53	00	/etc/init.d/rcS.
2:9100h:	2F	68	6F	6D	65	2F	68	69	6B	2F	73	74	61	72	74	2E	/home/hik/start.
2:9110h:	73	68	00	00	2F	65	74	63	2F	63	72	6F	6E	74	61	62	sh../etc/crontab
2:9120h:	00	00	00	00	2F	65	74	63	2F	63	72	6F	6E	2E	68	6Fetc/cron.ho
2:9130h:	75	72	6C	79	2F	78	00	00	2F	65	74	63	2F	72	63	2E	urly/x../etc/rc.
2:9140h:	64	2F	72	63	00	00	00	00	31	39	32	2E	31	36	38	2E	d/rc....192.168.
2:9150h:	31	2E	33	00	25	64	00	00	53	79	73	20	69	6E	69	74	1.3.%d..Sys init
2:9160h:	3A	20	4F	4B	00	00	00	00	2D	6E	6F	62	67	00	00	00	: OK....-nobg...
2:9170h:	4E	6F	20	73	79	6E	63	00	42	61	64	20	69	6E	69	74	No sync.Bad init
2:9180h:	00	00	00	00	25	64	20	25	64	20	25	64	0A	00	00	00%d %d %d....
2:9190h:	03	01	A8	C0	03	01	A8	C0	03	01	A8	C0	00	00	00	00	..`À..`À..`À....
2:91A0h:	2F	62	69	6E	2F	73	68	00	2D	63	00	00	65	78	69	74	/bin/sh.-c..exit
2:91B0h:	20	30	00	00	00	00	74	40	00	00	00	20	00	00	00	01	0....t@... ..

Linux/Moose in a Nutshell

- Affects routers / Internet of Things (IoT)
 - Embedded Linux systems with busybox userland
- Worm-like behavior
 - Telnet credential bruteforce
- Payload: Proxy service
 - SOCKSv4/v5, HTTP, HTTPS
- Used to proxy traffic to social media sites

Linux/Moose in a Nutshell

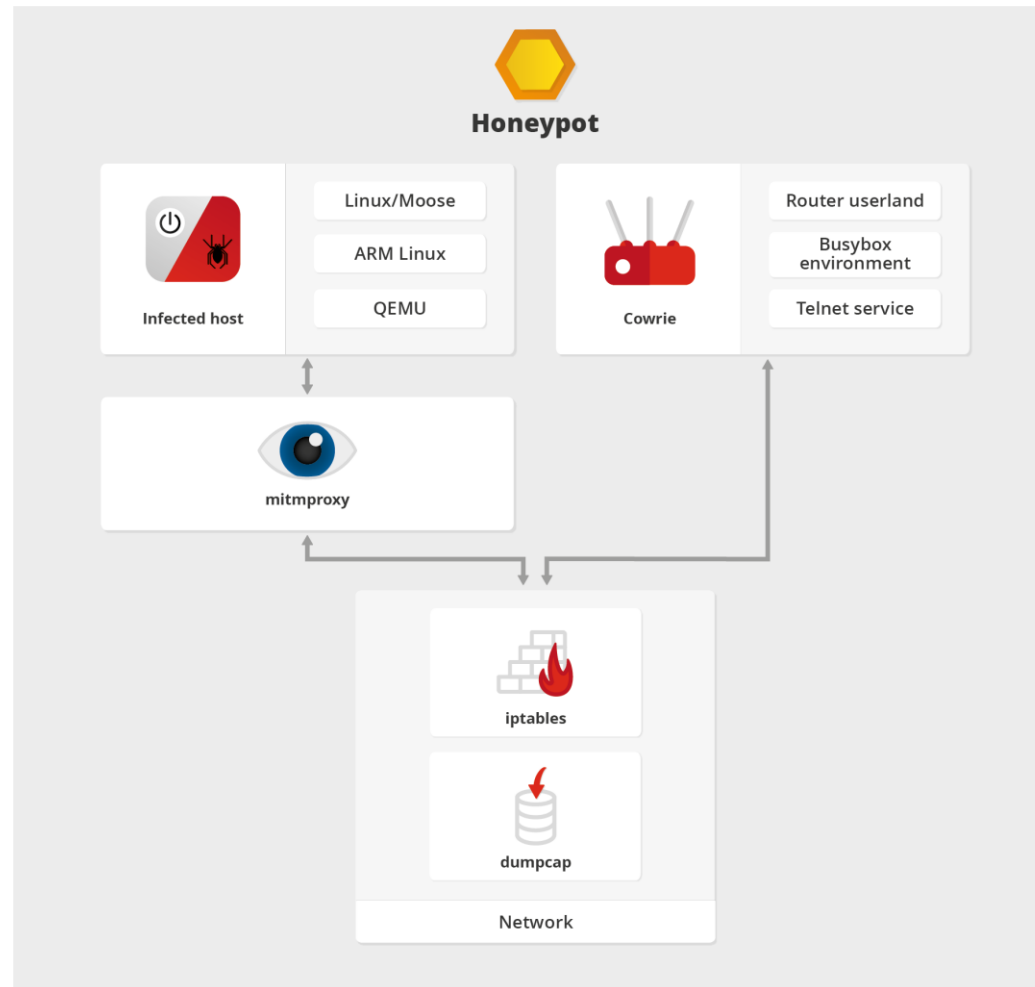
Timeline

- November 2014: Discovery by ESET
- Early 2015: Thoroughly reversed-engineered
- May 2015: Paper published
- June 2015: C&C down
- September 2015: New version
- Back then we decided to study it via detonation in special honeypots



Honeypots for Linux/Moose

Architecture of the Honeypot



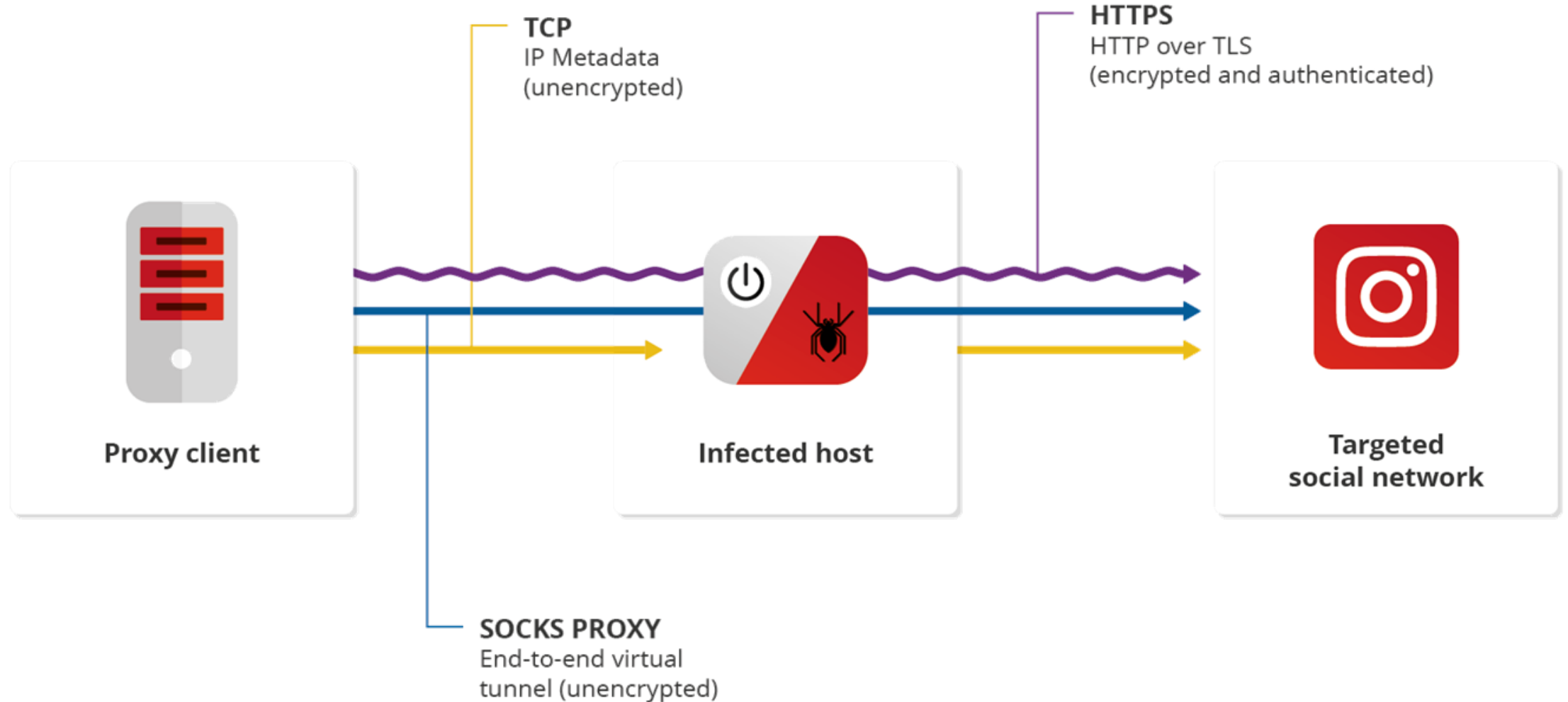
We set up honeypots all around the world



Today, we give you access to that network traffic!

Attacking Linux/Moose: Gaining access to its network traffic

How the Bots are Relaying Traffic





Workshop Time!

Dataset and Procedure

Dataset: a few days of the traffic gathered in the Singapore and Frankfurt honeypots!

- **Section 2 – Pcap analysis and data extraction**
 - Lab 0 – Introduction to Jupyter Notebook
 - Lab 1 – Explore with Wireshark and extract with Tshark (Wireshark's command-line interface)
 - Lab 2 – Scale Pcap data extraction with GNU parallel
- **Section 3 – Data manipulation and graphs**
 - Lab 3 – Manipulate dataframes with network traffic with Pandas
 - Lab 4 – Graph data using Plotly (hvplot)

But first!

Familiarize yourself with the environment!

<https://data-workshop.gosec.co>

TODO List

- ✓ Login (via your GitHub account)
- ✓ Open a Notebook
- ✓ Download a Pcap (in workshop/pcaps/)
- ✓ Optional: Download the slide deck (in workshop/)
- ✓ Optional: Open a Terminal

Once most of you connected, we will move on to the next section.

Pcap analysis and data extraction

Lab 0 - Intro to Jupyter Notebook

Jupyter Notebook

- Perform server-side computing with a Web UI
- Convenient Python and Shell bridge
- Data is in an environment ready for data analysis

Jupyter Notebook

- Run all the cells from the “labs/Lab 0 - Intro to Jupyter Notebook.ipynb” notebook

<https://data-workshop.gosec.co>

- Hurry here! The solution will be demonstrated shortly.
- Here are resources to learn more:
 - [How to Use Jupyter Notebook in 2020: A Beginner's Tutorial](#)
 - [Tutorial: Advanced Jupyter Notebooks](#)

Pcap analysis and data extraction

Lab 1 - Explore with Wireshark and scale with Tshark

Initial Packet Capture Analysis

A reminder of Wireshark's important features

- Protocol Hierarchy
- Conversations
- Follow TCP Stream
- Decode As
- Prepare a Filter

Important IP Addresses!

- Frankfurt 139.162.186.49
- Singapore 139.162.52.243

Your Assignment 1.1

- Linux/Moose is an IoT worm which attempts to self-replicate using Telnet, communicates with a C&C, and its payload is to proxy traffic.
- Open the moosehive_cn_01_00200... pcap in Wireshark and try to identify each type of traffic.
- <https://data-workshop.gosec.co>

Pcap files are in the workshop/pcaps/ directory of the Jupyter Notebook environment.

Automating Your Analysis with Tshark

- Tshark is the command-line tool that is part of the Wireshark suite. It uses the same engine and filtering language.

- To view all possible fields

```
tshark -G
```

- To extract to a CSV file (Tab separated)

```
tshark -T fields -e ip.src -e socks.dstport -r "pcapfile" -Y "(display filter)" > output.csv
```

- You can apply a “Decode As”

```
-d tcp.port==12345,socks
```

- Protip: Build your query using Wireshark then apply as a tshark display filter

Combining Tshark with Jupyter Notebook

- Create a more maintainable pcap data analysis pipeline than using a shell
 - It documents the commands in a durable form
 - And it is executable!

Other possible advantages include:

- You can avoid creating temporary files and load results directly in Python
- Perform the computing on server with more resources instead of your desktop
- Take advantage of data locality

Your Assignments 1.2, 1.3 and 1.4

- Run all the cells from the “labs/Lab 1 - Explore with Wireshark and extract with Tshark.ipynb” notebook
- Do it from our lab environment:
 - <https://data-workshop.gosec.co>

Hints for 1.2

- As seen in assignment 1.1, C&C check-ins are done over HTTP and the data is encoded in fake PHPSESSID cookies.
- You want to filter out as much of the unneeded packets as possible
 - Build a precise filter containing only the packets with the fields you want
- Human readable timestamps make it easier to deduce time deltas

Hints for 1.3

- To force a specific dissector to apply to a non-standard port (like “Decode As”) use:
 - `-d tcp.port==20012,socks`
- Only the first connect packet seems to have the proper “socks.dst” information (might be a wireshark bug)
 - `tcp.seq` can help you here

Hints for 1.4

- TLS before version 1.3 has plaintext metadata
 - You looked at the *Server Name Indication extension*?

1.1 Basic Exploration with Wireshark

Solution Demo

- Scanning behavior
- C&C
- Honeypot traffic
- Proxy traffic

Wireshark · Protocol Hierarchy Statistics · moosehive_cn_01_00200_20160828183344.pcap

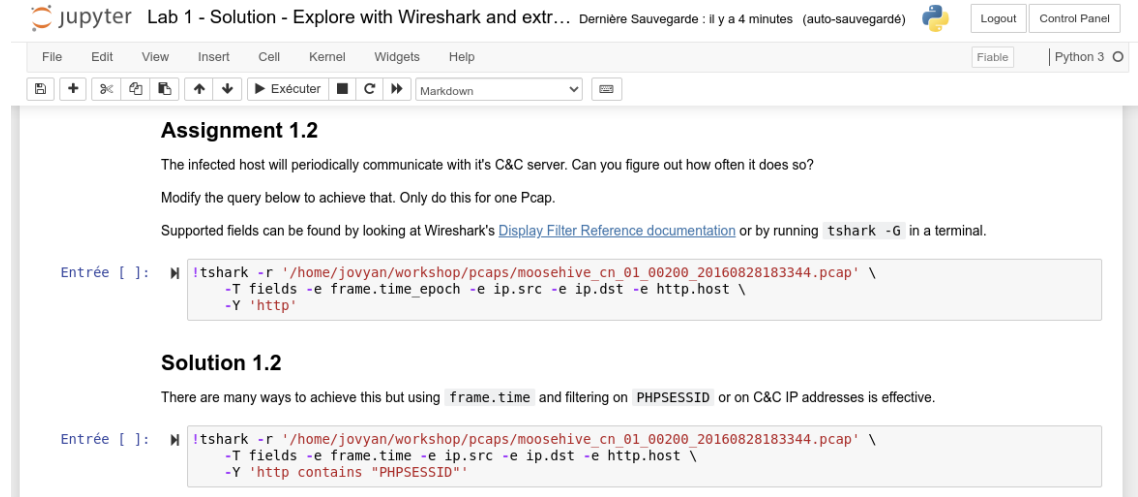
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	E
▼ Frame	100.0	224496	100.0	27008331	2,500	0	0
▼ Ethernet	100.0	224496	11.6	3142944	291	0	0
▼ Internet Protocol Version 6	1.3	2889	0.4	115560	10	0	0
Internet Control Message Protocol v6	1.3	2889	0.7	184896	17	2889	1
▼ Internet Protocol Version 4	96.7	217008	16.1	4340160	401	0	0
▶ User Datagram Protocol	0.3	644	0.0	5152	0	2	1
▼ Transmission Control Protocol	94.7	212616	69.1	18662506	1,728	192897	9
Transport Layer Security	2.4	5321	20.5	5550014	513	5108	4
Telnet	4.1	9257	3.4	904894	83	9257	9
▶ Socks Protocol	2.2	5000	8.6	2312731	214	1104	1
Malformed Packet	0.0	2	0.0	0	0	2	0
▼ Hypertext Transfer Protocol	0.2	362	1.3	337704	31	194	6
Line-based text data	0.1	168	0.1	29736	2	168	2
Data	0.0	2	0.0	3499	0	2	3
Internet Control Message Protocol	1.7	3748	0.9	234378	21	3748	2
Address Resolution Protocol	2.0	4599	0.6	169722	15	4599	1

1.2 Solution

Demo in the Jupyter Notebook!

Step by step

- `tshark -r "one pcap"`
- `-Y 'http contains "PHPSESSID"'`
- `-T fields -e frame.time -e ip.src -e ip.dst -e http.host`
 - Human readable timestamps



Assignment 1.2

The infected host will periodically communicate with its C&C server. Can you figure out how often it does so?

Modify the query below to achieve that. Only do this for one Pcap.

Supported fields can be found by looking at Wireshark's [Display Filter Reference documentation](#) or by running `tshark -G` in a terminal.

```
Entrée [ ]: tshark -r '/home/jovyan/workshop/pcaps/moosehive_cn_01_00200_20160828183344.pcap' \
-T fields -e frame.time_epoch -e ip.src -e ip.dst -e http.host \
-Y 'http'
```

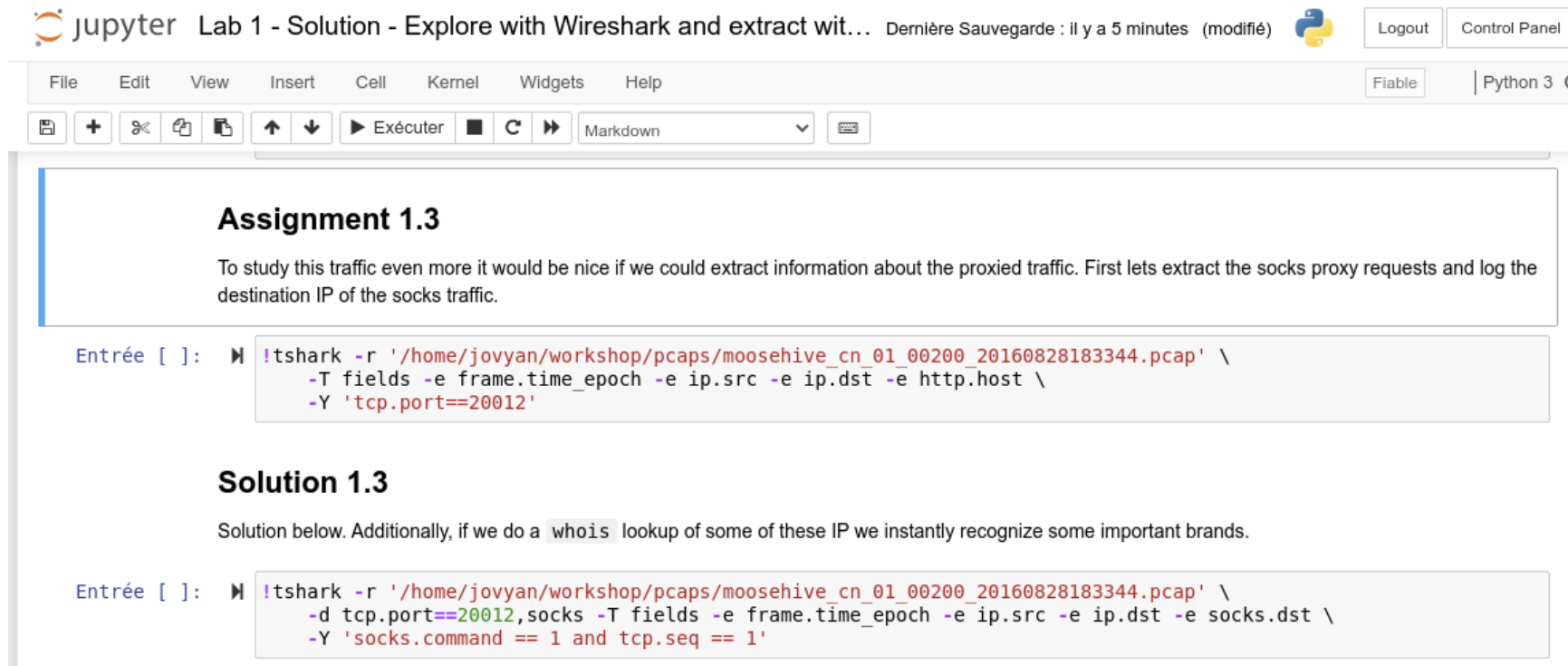
Solution 1.2

There are many ways to achieve this but using `frame.time` and filtering on `PHPSESSID` or on C&C IP addresses is effective.

```
Entrée [ ]: tshark -r '/home/jovyan/workshop/pcaps/moosehive_cn_01_00200_20160828183344.pcap' \
-T fields -e frame.time -e ip.src -e ip.dst -e http.host \
-Y 'http contains "PHPSESSID"'
```

1.3 Solution

- Demo in the Jupyter Notebook!



The screenshot shows a Jupyter Notebook titled "Lab 1 - Solution - Explore with Wireshark and extract wit...". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for saving, undo, redo, and execution, and a status bar indicating "Python 3".

Assignment 1.3

To study this traffic even more it would be nice if we could extract information about the proxied traffic. First lets extract the socks proxy requests and log the destination IP of the socks traffic.

Entrée []: ▶ `!tshark -r '/home/jovyan/workshop/pcaps/moosehive_cn_01_00200_20160828183344.pcap' \`
`-T fields -e frame.time_epoch -e ip.src -e ip.dst -e http.host \`
`-Y 'tcp.port==20012'`

Solution 1.3

Solution below. Additionally, if we do a `whois` lookup of some of these IP we instantly recognize some important brands.

Entrée []: ▶ `!tshark -r '/home/jovyan/workshop/pcaps/moosehive_cn_01_00200_20160828183344.pcap' \`
`-d tcp.port==20012,socks -T fields -e frame.time_epoch -e ip.src -e ip.dst -e socks.dst \`
`-Y 'socks.command == 1 and tcp.seq == 1'`

1.4 Solution

- Demo in the Jupyter Notebook!

jupyter Lab 1 - Solution - Explore with Wireshark and extract wit... Dernière Sauvegarde : il y a 5 minutes (modifié) Logout Control Panel

File Edit View Insert Cell Kernel Widgets Help Fiable Python 3

Exécuter

```
-d tcp.port==20012,socks -T fields -e frame.time_epoch -e ip.src -e ip.dst -e socks.dst \
-Y 'socks.command == 1 and tcp.seq == 1'
```

Assignment 1.4

Can we extract TLS metadata that would allow us to get a more precise understanding of the intended traffic target?

Entrée []: `!tshark -r '/home/jovyan/workshop/pcaps/moosehive_cn_01_00200_20160828183344.pcap' \
-T fields -e frame.time_epoch -e ip.src -e ip.dst \
-Y 'tls'`

Solution 1.4

Filter on TLS client handshake (`tls.handshake.type == 1`) and extract server name extension (`tls.handshake.extensions_server_name`):

Entrée []: `!tshark -r '/home/jovyan/workshop/pcaps/moosehive_cn_01_00200_20160828183344.pcap' \
-T fields -e frame.time_epoch -e ip.src -e ip.dst -e tls.handshake.extensions_server_name \
-Y 'tls.handshake.type == 1'`

Bonus

Here is some filtering code to keep unique domains. Just fix the `tshark` command with the right parameters from your last solution.

Entrée []: `proxy_dest = !tshark -r '/home/jovyan/workshop/pcaps/moosehive_cn_01_00200_20160828183344.pcap' \
-T fields -e frame.time_epoch -e ip.src -e ip.dst \
-Y 'tls'`

```
proxy_dest_domains = {}
for _dst in proxy_dest:
    epoch, srcip, dstip, domain = _dst.split("\t")
    # fill dict with domain name as key, duplicates will overwrite each other
    proxy_dest_domains[domain] = 1

# turn dict into a list
domains = [key for key, val in proxy_dest_domains.items()]
sorted(domains)
```

Pcap analysis and data extraction

Lab 2 – Scale Pcap data extraction with GNU parallel

Group Assignment 2.1

- Together we will run all the cells from the “labs/Lab 2 - Scale Pcap data extraction with GNU parallel.ipynb” notebook

jupyter Lab 2 - Scale Pcap data extraction with GNU parallel (auto-sauvegardé)

File Edit View Insert Cell Kernel Widgets Help

Exécuter

Multi-core xargs: GNU parallel

Can we go faster? Yes by spreading the `tshark` jobs on several cores using GNU parallel. Below is the solution. It is bui and failures using it.

```
Entrée [2]: %%time

socks_dest = !find "/home/jovyan/workshop/pcaps/" -name "*.pcap" | \
parallel --max-args 1 --jobs 80% --load 100% --memfree 2G --retries 3 --q
tshark -d tcp.port==20012,socks -r \{\} -T fields -e frame.time_epoch -e
-Y 'socks.command == 1 and tcp.seq == 1'

tls_dest = !find "/home/jovyan/workshop/pcaps/" -name "*.pcap" | \
parallel --max-args 1 --jobs 80% --load 100% --memfree 2G --retries 3 --q
tshark -r \{\} -T fields -e frame.time_epoch -e ip.src -e ip.dst -e tls.h
-Y 'tls.handshake.type == 1'
```

CPU times: user 6.05 ms, sys: 12.1 ms, total: 18.1 ms
Wall time: 1min 32s

```
Entrée [3]: # Write files to disk
with open('/home/jovyan/work/socks-handshake.csv', 'w') as _f:
    for line in socks_dest:
        _f.write(line + "\n")

with open('/home/jovyan/work/tls-handshake.csv', 'w') as _f:
    for line in tls_dest:
        _f.write(line + "\n")
```

- Do it yourself from our lab environment:

– <https://data-workshop.gosec.co>

Data manipulation and graphs

Lab 3 – Manipulate dataframes with network traffic using Pandas

Lab 3 – Manipulate dataframes with network traffic with Pandas

What is Pandas?

“an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.”



Lab 3 – Manipulate dataframes with network traffic with Pandas

1. We will load two CSVs created with tshark
2. We will merge the two CSVs and create one large clean dataframe
3. I will show a few tricks using Pandas
4. You will be given some questions to answer!

Useful documentation:

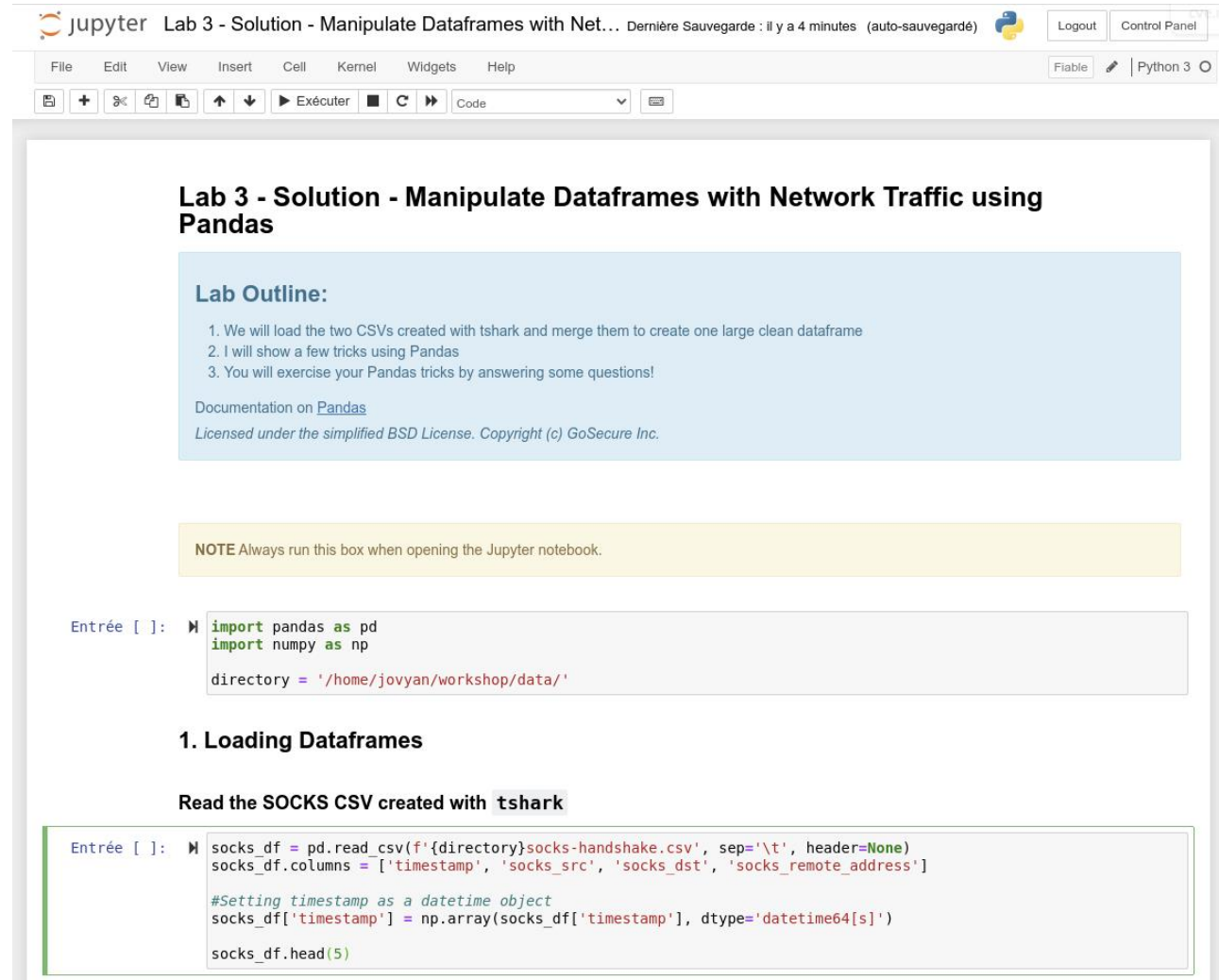
<https://pandas.pydata.org/>



Lab 3 - Manipulate Dataframes with Network Traffic using Pandas

Solution

Demo and a Jupyter notebook including the lab's solution will be provided in the participants' virtual environments



The screenshot shows a JupyterLab interface with a notebook titled "Lab 3 - Solution - Manipulate Dataframes with Network Traffic using Pandas". The interface includes a top bar with the Jupyter logo, the notebook title, and a "Dernière Sauvegarde : il y a 4 minutes (auto-sauvegardé)" status. Below the top bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar below the menu bar contains icons for file operations, a "Code" dropdown, and a "Python 3" kernel selector. The notebook content is displayed in a light blue box with the title "Lab 3 - Solution - Manipulate Dataframes with Network Traffic using Pandas". Below the title is a "Lab Outline:" section with three numbered steps: 1. We will load the two CSVs created with tshark and merge them to create one large clean dataframe. 2. I will show a few tricks using Pandas. 3. You will exercise your Pandas tricks by answering some questions! Below the outline is a link to "Documentation on Pandas" and a note: "Licensed under the simplified BSD License. Copyright (c) GoSecure Inc." Below the outline is a yellow box with the text: "NOTE Always run this box when opening the Jupyter notebook." Below the note is a code cell with the following code:

```
Entrée [ ]: In import pandas as pd
import numpy as np

directory = '/home/jovyan/workshop/data/'
```

 Below the code cell is a section titled "1. Loading Dataframes" with a sub-section titled "Read the SOCKS CSV created with tshark". Below the sub-section is a code cell with the following code:

```
Entrée [ ]: In socks_df = pd.read_csv(f'{directory}socks-handshake.csv', sep='\t', header=None)
socks_df.columns = ['timestamp', 'socks_src', 'socks_dst', 'socks_remote_address']

#Setting timestamp as a datetime object
socks_df['timestamp'] = np.array(socks_df['timestamp'], dtype='datetime64[s]')

socks_df.head(5)
```

Data manipulation and graphs

Lab 4 - Graph data using hvplot

Lab 4 - Graph data using hvplot

Lab:

1. We will go over a few tricks with hvplot
2. You will exercise with a few challenges!

Useful documentation:

<https://hvplot.holoviz.org/>

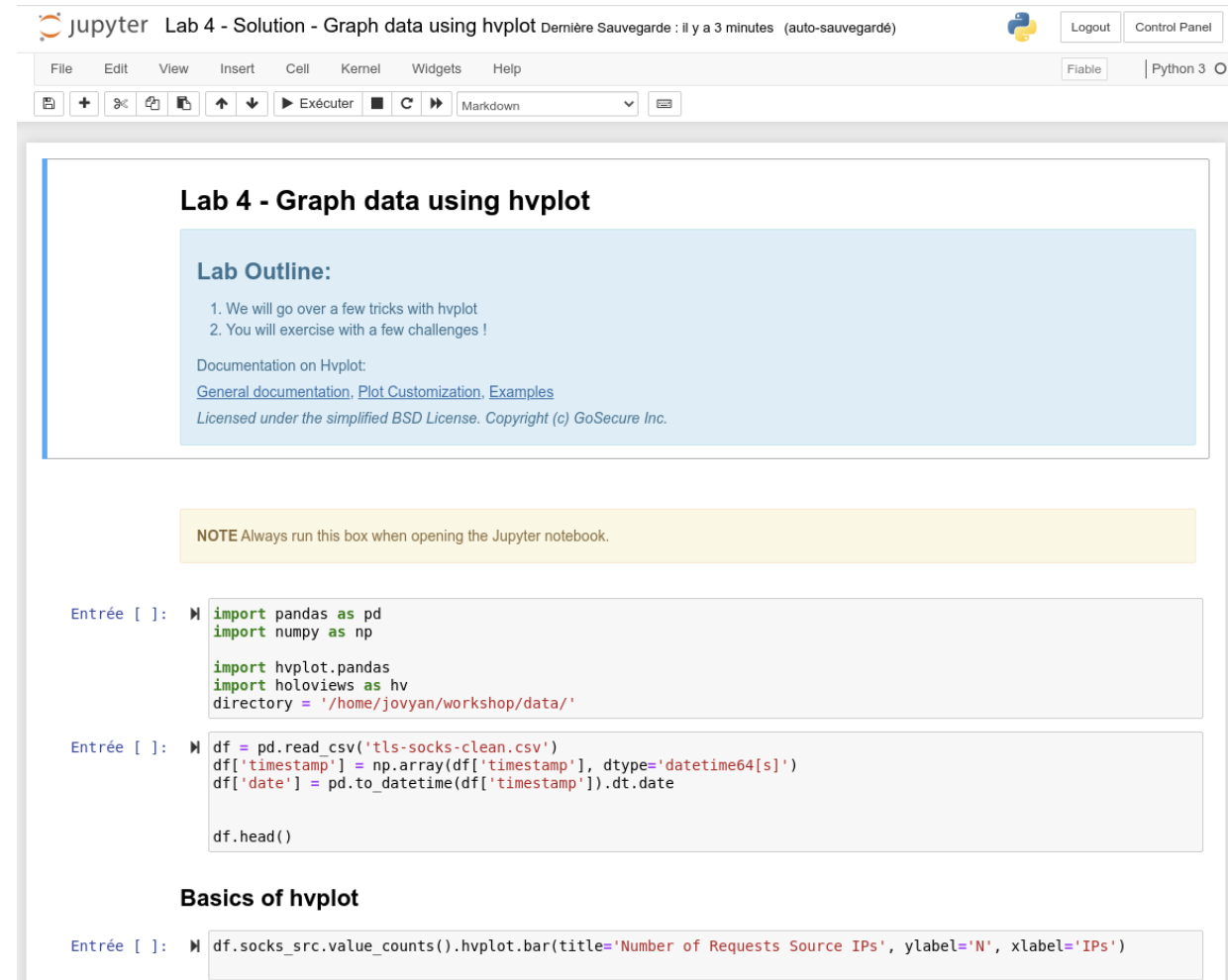
http://holoviews.org/user_guide/Customizing_Plots.html

https://hvplot.holoviz.org/user_guide/Customization.html

Lab 4 - Graph data using hvplot

Solution

Demo and a Jupyter notebook including the lab's solution will be provided in the participants' virtual environment



The screenshot displays a JupyterLab environment. At the top, the title bar reads 'jupyter Lab 4 - Solution - Graph data using hvplot' with a timestamp 'Dernière Sauvegarde : il y a 3 minutes (auto-sauvegardé)' and buttons for 'Logout' and 'Control Panel'. Below this is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. A toolbar contains icons for file operations and execution, with a dropdown menu currently set to 'Markdown'. The main workspace area is titled 'Lab 4 - Graph data using hvplot' and contains a 'Lab Outline' section with two points: '1. We will go over a few tricks with hvplot' and '2. You will exercise with a few challenges!'. It also includes links to 'Documentation on Hvplot: General documentation, Plot Customization, Examples' and a license notice: 'Licensed under the simplified BSD License. Copyright (c) GoSecure Inc.'. A yellow note box states: 'NOTE Always run this box when opening the Jupyter notebook.' Below this are two code input areas. The first, labeled 'Entrée []:', contains code to import pandas, numpy, hvplot.pandas, and holoviews, and to set a directory. The second, also labeled 'Entrée []:', contains code to read a CSV file, convert timestamps to dates, and display the first few rows. At the bottom, a section titled 'Basics of hvplot' shows a code input area with a single line of code to create a bar plot from the 'df' dataframe.

Lab 4 - Graph data using hvplot

Lab Outline:

1. We will go over a few tricks with hvplot
2. You will exercise with a few challenges !

Documentation on Hvplot:
[General documentation](#), [Plot Customization](#), [Examples](#)
Licensed under the simplified BSD License. Copyright (c) GoSecure Inc.

NOTE Always run this box when opening the Jupyter notebook.

```
Entrée [ ]: ▶ import pandas as pd
import numpy as np

import hvplot.pandas
import holoviews as hv
directory = '/home/jovyan/workshop/data/'

Entrée [ ]: ▶ df = pd.read_csv('tls-socks-clean.csv')
df['timestamp'] = np.array(df['timestamp'], dtype='datetime64[s]')
df['date'] = pd.to_datetime(df['timestamp']).dt.date

df.head()
```

Basics of hvplot

```
Entrée [ ]: ▶ df.socks_src.value_counts().hvplot.bar(title='Number of Requests Source IPs', ylabel='N', xlabel='IPs')
```

Lab Key Takeaways

We hope that through this lab, you have learned:

- How to use Wireshark and Tshark to extract meaningful information from malicious network traffic captures
- How to scale and self-document traffic extraction by combining Jupyter Notebooks and Tshark
- How to efficiently graph the extracted traffic to highlight hidden patterns using open-source Python libraries
- Network traffic investigation skills 😊

One Last Word About Scalability

- What you have learned today could have been done with Wireshark and Excel
 - It's 21 days of honeypot traffic representing 3.9M packets
- Our Linux/Moose investigation ran approximately 10 honeypots for 6 months gathering more than 2700 days of traffic
 - That's approximately 500M packets
 - Enough to blow up Excel and a laptop's RAM and require the more robust and scalable tools you saw in action today



LAB3-T07

Malicious Network Data Analysis Using Open-Source Tools

Olivier Bilodeau

Cybersecurity Research Director

GoSecure

@obilodeau

#GOSEC