THALES

BRETAGNE
A network simulator for AI

BRETAGNE

Building a Reproducible and Efficient Training AI Gym for Network Environments

Python Documentation

-Thomas Lefeuvre
-Yann Gourlet

Secure Communications & Information (SIX), Thales Deutschland, Ditzingen, Germany
Ecole Supérieure d'Ingénieurs de Rennes (ESIR), University of Rennes, Rennes, France

We certify that this work is original, that it appropriately indicates all borrowings, and that it appropriately references each source used.

Table des matières

1 Introduction

BRETAGNE, is a network simulation environment designed to serve as a training ground for autonomous defense agents using hybrid AI models in simulations. The platform integrates docker, a lightweight virtualization technology orchestrated by Kathara with widely used network protocols such as BGP, OSPF, HTTP, SSH and others to simulate production-like environments. We present a multi-agent architecture involving blue, red, green, and white agents, designed to create dynamic, communicative environments for training purposes. Overall, the BRETAGNE framework offers a realistic and scalable solution for the training and deployment of autonomous agents in operational networks.

1.1 Key Features

- Realism : Using docker to simulate an environment with real implementations.

- Scalability : Create your own network scenario quickly with built-in python functions.

- Performant : Using containerization to simulate large networks.

- Autonomous trafic : Green and red agent implementations for realistic, autonomous network traffic generation.

- Interaction : Each machine can be operated during simulation via its own terminal.

- Automatic monitoring : Automatic monitoring of attacks using LLM and decision-making using SDM.

2 Installation

1. To install bretagne, start by downloading the install.sh script:
https://github.com/ThomasL53/BRETAGNE/blob/main/install.sh

2. Move your file to your home directory and give it installation rights:
```
sudo chmod +x install.sh
```

3. Run the installation script (This may take some time depending on your internet connection). If an error occurs, refer to /tmp/InstallBretagne.log
```
./install.sh
```

4. Reboot your computer or close your terminal to finalize installation

5. Don't forget to install AWS CLI and configure your login with 'AWS configure' to use the blue agent.

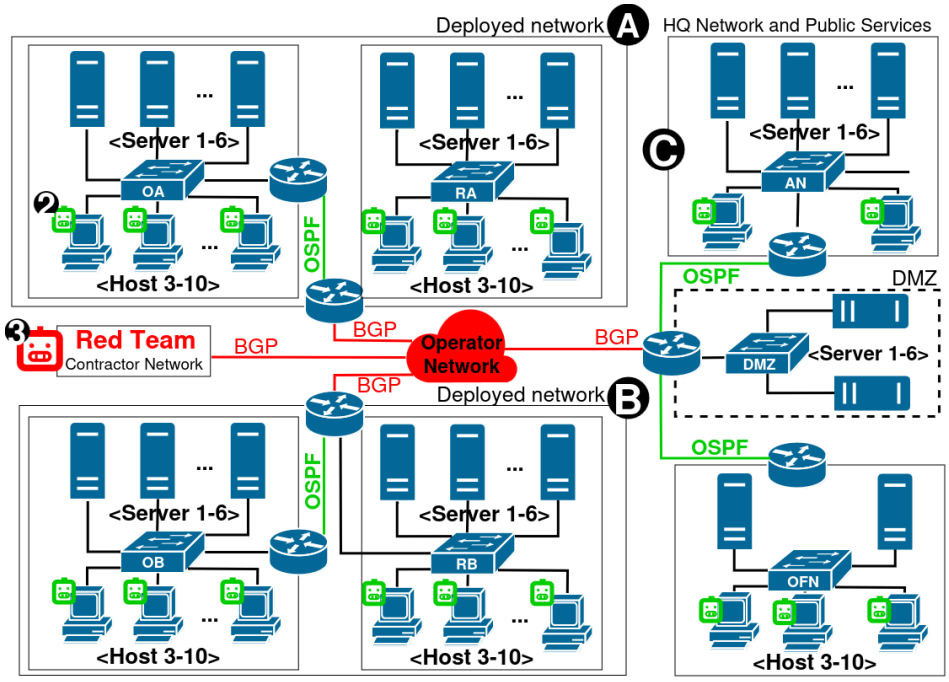
6. Please note that the initial start-up of the simulation may take some time, depending on the images already present on your machine.

2.1 Example of use

Starting a simulation with metasploit on the Operator Network (ON) and on the network Restricted Zone A (RA):
`bretagne --start --metasploit ON RA`
Open a terminal on pc_ra1:
`bretagne --control pc_ra1`
Observe traffic on the Operator Network (ON):
`bretagne --monitor ON`
Generating user traffic on the network:
`bretagne --generate_traffic 10`
Deploy a blue agent on the Operator Network (ON):
`bretagne --BlueAgent ON`
Stop the simulation:
`bretagne –stop`
For more help:
`bretagne -h`

3 Default Topology

The default network is that used in Cage Challenge 4. The network consists of two theater networks, shown as 'A' and 'B' and a metropolitan network, shown as 'C', interconnected by a traditional carrier network (ON). In addition, a corporate network (CN) is connected to the same operator.The subnetworks used in the simulation are specified on the schematic switch.



- **The routers.** Utilize FRR 9.0.1 and it's name fw_'NETWORK name' (exemple: for the RA network the router is fw_ra)

-**The switches**. Employ Open vSwitch 3.0.1 and it's name ovs_'NETWORK name' (exemple: for the OFN network the switch is ovs_ofn). They are all connected to the SDN controller via a common control plane.

- **The hosts.** Are randomly generated between 3 and 10 on each subnetwork. The are are based on a Linux kernel 6.8.0-39 and it's name pc'x'_'NETWORK name' (exemple: for the DMZ network the host 1 is pc1_dmz)

- **The server**. Are randomly generated between 1 and 6 on each subnetwork. The are are based on a Linux kernel 6.8.0-39 and it's name srv'x'_'NETWORK name' (exemple: for the RB network the server 3 is srv3_dmz)

- **The SDN controler**. Is based on Floodlight 1.2. It is accessible via the URL:http://localhost:8080/ui/pages/index.html. It's name 'controller'

4 Developer Guide

For a better understanding of the BRETAGNE source code, it is highly recommended to refer to or rely on the documentation of the Kathara Python API: https://github.com/KatharaFramework/Kathara/wiki/Kathara-API-Docs

4.1 Starting and initializing a simulation

BRETAGNE is usable via the command line. This command line is made accessible through the installation program, which adds an ALIAS to the machine's bashrc, or via the env.sh script which, once sourced, provides access to the environment. The bretagne.py file at the root of the project allows the creation of this command-line interface and orchestrates each function of BRETAGNE.

This file checks if the current working directory contains the word 'BRETAGNE'. If it does not, it displays an error message and exits the program :

```
def main(args):
    if "BRETAGNE" not in os.getcwd():
        print(f"{RED}You must execute this command in the BRETAGNE working directory!{RESET}")
        quit()
```

"It creates a working directory named 'simu' if it does not already exist when starting a simulation. This directory also serves as an indicator of the simulation start.

```
if not os.path.exists("simu") and args.start:
    os.makedirs("simu")
elif os.path.exists("simu"):
    start = True
```

Based on the command-line arguments provided, it executes various actions such as starting the simulation, deploying agents (BlueAgent, WhiteAgent), controlling the network, generating traffic, monitoring the network, evaluating networks, etc.

The creation and start of the simulation are carried out in the file BRETAGNE/init.py. The function create_network orchestrates the creation of this simulation. Initially, each router is added to the scenario using the function BRETAGNE/utils/Sim_tools/add_router_frr.

```python
#Function for add a fr routing router
def add_router_frr(name,lab):
    router=lab.new_machine(name.lower(), **{"image": "kathara/frr"})
    lab.create_file_from_list(
        [
            "/etc/init.d/frr start",
        ],
        f"{name}.startup"
    )
    return router
```

This function utilizes the Kathara API to create a router based on the Docker image FRRouting 9.0.1. The IP configuration of the routers is then added using the function BRETAGNE/utils/Sim_tools/add_ip_addr_on.

```python
def add_ip_addr_on(machine,eth,ip,lab):
    if os.path.isfile(f"simu/{machine}.startup"):
        lab.update_file_from_list(
        [
            f"/sbin/ifconfig {eth} {ip} up",
        ],
        f"{machine}.startup"
        )
    else:
        lab.create_file_from_list(
        [
            f"/sbin/ifconfig {eth} {ip} up",
        ],
        f"{machine}.startup"
        )
```

This function adds the ifconfig commands to the .startup file of the machine using the Kathara API to configure an IP interface. (Each command in this .startup file will be executed upon machine startup). Finally, the configuration of the FRR routers (BGP, OSPF, etc.) is pushed to the routers using the function BRETAGNE/utils/Sim_tools/configure_frr_on :

```python
def configure_frr_on(router):
    router.create_file_from_path(os.path.join("config", f"{router.name}.conf"), "/etc/frr/frr.conf")
    router.create_file_from_path(os.path.join("config", "daemons"), "/etc/frr/daemons")
    router.create_file_from_string(content="service integrated-vtysh-config\n", dst_path="/etc/frr/vtysh.conf")
    router.update_file_from_string(content=f"hostname {router.name}\n", dst_path="/etc/frr/vtysh.conf")
```

This program will use the Kathara API to share the FRR configuration files (available in the config directory) by using a shared directory mounted on the Docker container.

The continuation of the scenario creation function will initialize the different subnets using the function BRETAGNE/init/create_subnet :

```
create_subnet("RA",lab,subnet_count,"1.1.1.0")
```

This function takes parameters such as a name, the Kathara working lab, the number of already instantiated subnets, and a network address in /24. The function then generates a random number of hosts between 3 and 10, a number of servers between 1 and 6, and a switch based on the Docker image Open vSwitch 3.0.1. The switch will interconnect each node in the subnet. The default gateway address is also generated based on the provided /24 addressing.

```
def create_subnet(name,lab,subnet_count,subnet_addr=None):
    eth=1
    name=name.lower()
    #choice of number of subnet hosts
    nb_PC=random.randint(3, 10)
    PC_list=[]
    #choice of number of subnet serveurs
    nb_SRV=random.randint(1, 6)
    SRV_list=[]
    #creation of an IPv4 object
    network = ipaddress.IPv4Network(subnet_addr)
    ips = []
    #Creation of the network address (by definition, the last address in the range)
    gateway=".".join(subnet_addr.rsplit(".", 1)[:-1] + ["254"])
    #copy packages to machines
    BRETAGNE.utils.Sim_tools.add_package()
    OvS=BRETAGNE.utils.Sim_tools.add_SDN_switch(name,subnet_count,lab)
```

Each host in the simulation is named pc_<network name>_x and is connected to its switch via a "collision domain" using the Kathara API.

```
for pc in range(1,nb_PC+1):
    PC_name="pc_" + name + str(pc)
    PC = lab.new_machine(PC_name)
    #connecting PCs to the switch
    lab.connect_machine_to_link(PC.name,(name+PC.name).upper())
    lab.connect_machine_to_link(OvS.name,(name+PC.name).upper())
    lab.update_file_from_list(
        [
            f"ovs-vsctl add-port s1 eth{eth}"
        ],
        f"ovs_{name}.startup"
    )
    eth=eth+1
    PC_list.append(PC)
```

The creation of servers follows the same logic, but an additional operation is performed to configure the server services using the function BRETAGNE/utils/Sim_tools/add_srv_service_on :

```python
#Function for add a predefined service to a node
def add_srv_service_on(SRV,lab):
    lab.create_file_from_list(
        [
            "/etc/init.d/apache2 start",
            "apt install ./shared/packages/vsftpd.deb",
            "/etc/init.d/vsftpd start",
            "/etc/init.d/ssh start",
            "/etc/init.d/named start",
            "/etc/init.d/bind start"
        ],
    f"{SRV}.startup"
    )
```

This function uses the Kathara API to add the startup and installation of services to the .startup file if necessary. The rest of the function utilizes the Kathara API to perform the IP configuration of the machines by adding an IP and the default gateway. The IPs are distributed sequentially starting from .1. The IPs of all servers are recorded in the file simu/srv_iplist.

```python
#Creating IPs for machines
for i in range(1,nb_PC+nb_SRV+1):
    ip = str(network.network_address + i)
    ips.append(ip)

machines= PC_list + SRV_list

#Machine IP configuration
for i, machine in enumerate(machines):
    lab.update_file_from_list(
        [
            f"/sbin/ifconfig eth0 {ips[i]}/24 up",
            f"route add default gw {gateway}",
            "apt install ./shared/packages/ftp.deb"
        ],
    f"{machine.name}.startup"
    )
    #Create a file with server IPs and configure SRV service
    if "srv" in machine.name:
        file_path="simu/srv_iplist"
        if os.path.exists(file_path):
            with open(file_path, "a") as file:
                file.write( ips[i] + "\n")
        else:
            with open(file_path, "w") as file:
                file.write( ips[i] + "\n")
```

The different subnets are then connected to their routers using the "kathara" collision domains, and the script for sniffing network traffic on the subnets is deployed using the function BRETAGNE/utils/Sim_tools/add_monitoring.

```python
def add_monitoring(network,lab):
    nbport = count_port(f"ovs_{network.lower()}")
    lab.connect_machine_to_link(f"ovs_{network.lower()}","MNT")
    lab.update_file_from_list(
        [
         f"ovs-vsctl add-port s1 eth{nbport+1} -- --id=@p get port
        ],
    f"ovs_{network.lower()}.startup"
    )
    os.makedirs(f"simu/shared/script", exist_ok=True)
    os.makedirs(f"simu/shared/capture", exist_ok=True)
    src_file = f"script/snif.sh"
    dst_file = f"simu/shared/script/snif.sh"
    shutil.copy(src_file, dst_file)
```

This function utilizes the Kathara API to deploy the OvS command to set up port monitoring on each port of the subnet's switch and deploys a script on the switch container to orchestrate TCPDUMP for recording the traffic in a shared directory with the host machine of the simulation.

Also found in the init.py file, the start function orchestrates the startup of the simulation using the Kathara API, manages exceptions, and handles display using yaspin :

```python
def start(lab):
    with open("simu/labhash", "w") as file:
        file.write(lab.hash)
    with yaspin(Spinners.dots, text="Starting the simulation...") as spinner:
        try:
            Kathara.get_instance().deploy_lab(lab)
            spinner.text = ""
            spinner.ok("✓ Simulation started successfully!")
        except Exception as e:
            spinner.text = "✗ Simulation start failed!"
            spinner.fail("✗ Simulation start failed!")
            print(f"Error starting the simulation: {e}")
```

4.2 Interaction with machines

Once the simulation is running, there are various ways to interact with it. You can, for instance, open a terminal on each machine using the command --control. This command takes a machine name as a parameter and utilizes the BRETAGNE/control function that uses the Kathara API to open a terminal.

```python
#Open a terminal on the specified node
def control(name):
    print(f"Opening a console on {name}")
    Kathara.get_instance().connect_tty(name, lab_name="simu")
```

Each machine utilizes a real implementation of Linux. Therefore, it is possible to execute any command on these terminals (installation, launching programs, etc.).

It is also possible to observe network traffic with the command --monitoring, which will launch the tcpdump script on the specified network or networks provided in the command line :

```
from Kathara.manager.Kathara import Kathara

#This function launches TCPDUMP on the specified network with the dedicated script
def monitor(network):
    Kathara.get_instance().exec(f"ovs_{network}"," chmod 777 /shared/script/snif.sh",lab_name="simu")
    Kathara.get_instance().exec(f"ovs_{network}","./shared/script/snif.sh",lab_name="simu")
```

A PCAP file is then generated and updated in the simu/capture directory.

Finally, it is also possible to interact with the simulation through the SDN Floodlight controller at the address.: http://localhost:8080/ui/pages/index.html. To learn more about this interface, refer to this page: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview

This SDN interface can also be used via its API. Some examples are available in the file BRETAGNE/utils/SDN_action. These actions are utilized in Bretagne by the blue agent to block or allow traffic. Here is an example for allowing traffic:

```
def AllowTraffic(src_ip, dst_ip):
    url = f"http://localhost:8080/wm/acl/rules/json"
    headers = {"Content-Type": "application/json"}
    try:
        #recovery of ACLs
        response = requests.get(url)
        response.raise_for_status()
        acls = response.json()
        for acl in acls:
            #if an ACL blocking traffic exists, it is deleted
            if acl["nw_src"] == src_ip and acl["nw_dst"] == dst_ip and acl["action"] == "DENY":
                # Supprimer l'ACL
                acl_id = acl["id"]
                data = {"ruleid": acl_id}
                delete_response = requests.delete(url, data=json.dumps(data), headers=headers)
                delete_response.raise_for_status()
                return
        data = {
            "src-ip": src_ip,
            "dst-ip": dst_ip,
            "action": "allow"
        }
        #send request for creation of traffic allowing ACL
        try:
            response = requests.post(url, data=json.dumps(data), headers=headers)
            response.raise_for_status()
```

4.3 Green, red, white and blue agent

4.3.1 Green agent

BRETAGNE includes the ability to observe the operation of an LLM on the network, or to autonomously train an artificial intelligence model. To do this, a green agent can be deployed in the simulation using the command --Generate_traffic. The start function in the file BRETAGNE/Generate_traffic will orchestrate this traffic generation. To achieve this, a machine and a server are randomly selected from the simulation. Then, an action is randomly chosen from the available action list (DHCP, DNS, ping, ssh, web). This can be easily extended by adding the definition of your Python function in the file.

```python
def start(nb_iterations=20):
    actions= [generate_dhcp,generate_dns,generate_ping,generate_ssh,generate_www]
    global machine_list
    global ip_addresses
    global ip_srvlist

    machine_list = get_machine_names(directory)
    ip_addresses = get_ip_addresses(directory)
    ip_srvlist = get_srv_ip()
    for _ in range(nb_iterations):
        random_action = random.choice(actions)
        random_nb_connexion = random.randint(1, 20)

        random_action(random_nb_connexion)

        #random sleep 1s to 5s
        temps_attente = random.uniform(1, 5)
        time.sleep(temps_attente)
```

Here is an example for the web connection. The program uses the Katahra API to send a `wget` command from the randomly selected machine to the randomly selected server. This command executed on the machine will generate real traffic on the network as if a user had actually visited the web page.

```python
def generate_www(nbconnexion):
    for i in range(nbconnexion):
        Kathara.get_instance().exec(machine_name=random.choice(machine_list),command=f'wget {random.choice(ip_srvlist)}',wait=True, lab_name="simu")
        time.sleep(0.05)
```

All the IP addresses in the simulation are retrieved by scanning the `.startup` files of the machines and searching for the regex of an IPv4 address.

```python
#This function returns a list of all allocated IP addresses for this simulation
def get_ip_addresses(directory):
    ip_addresses = []
    for filename in os.listdir(directory):
        if filename.endswith(".startup"):
            file_path = os.path.join(directory, filename)
            with open(file_path, "r") as file:
                content = file.read()
                ip_match = re.search(r"/sbin/ifconfig eth\d (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/\d{1,2} up", content)
                if ip_match:
                    ip_address = ip_match.group(1)
                    ip_addresses.append(ip_address)
    return ip_addresses
```

4.3.2 Red Agent

Regarding the red agent, it can only be deployed on a network at the start of the simulation. Indeed, it is complicated with Kathara to connect a new machine during the simulation. This red agent uses a Metasploit Docker image in our tool. The deployment of this container is similar to that of the routers:

```
#Function for deploying metasploit on a network
def add_metasploit_on(network,lab):
    nbport = count_port(f"ovs_{network.lower()}")
    print(f"Add metasploit_{network.lower()} to network {network.upper()}")
    redagent=lab.new_machine(f"metasploit_{network.lower()}", **{"image": "metasploitframework/metasploit-framework"})
    lab.connect_machine_to_link(redagent.name, f"META{network}")
    lab.connect_machine_to_link(f"ovs_{network.lower()}",f"META{network}")
    lab.update_file_from_list(
        [
         f"ovs-vsctl add-port s1 eth{nbport+1}"
        ],
    f"ovs_{network.lower()}.startup"
    )
    os.makedirs(f"simu/shared/script", exist_ok=True)
    src_file = f"script/password"
    dst_file = f"simu/shared/script/password"
    shutil.copy(src_file, dst_file)
```

The main actions here are the creation of the machine, the connection of the machine to the switch, and the transfer of the file that will enable the launch of brute force attacks.

The attacks can then be launched in different ways. Manually, by opening a terminal on the red agent named "metasploit_<network_name>" and launching the `msfconsole`. Alternatively, they can be orchestrated using the white agent.

4.3.3 White agent

The white agent can be used in different ways. The first allows for the generation of a realistic and labeled dataset for training an artificial intelligence model. This functionality will use the monitoring available in BRETAGNE to record network traffic and will then randomly choose whether an attack occurs on the network or if a regular user makes a request:

```
#This function generate a CSV file with: the network traffic, the attacker IP, the victim IP amd t
def generate_dataset(network):
    pcap_file = f"simu/shared/capture/ovs_{network.lower()}.pcap"
    csv_file = f"simu/shared/capture/ovs_{network.lower()}.csv"
    dataset_file= f"dataset_{network.lower()}.csv"
    print(f"Generation of a dataset {dataset_file}. \n ctrl + c to stop the generation")
    while 1:
        BRETAGNE.utils.tools.clean_file(pcap_file,csv_file)
        BRETAGNE.monitoring.monitor(network)
        attack = random.randint(0,1)
        userTraffic = random.randint(0,1)
```

If an attack occurs, the random attack function will choose the type of attack as well as the IPs of the attacker and defender randomly, and it will execute the attack on the network. At the end of this attack, the program will return the IPs used and the executed attack. This information will enable the labeling of the data. If user traffic is executed, the function directly calls the green agent to perform user actions on the network. Finally, all this data is then added to a CSV file, including the captured traffic,

an attack flag, the IP of the attacker, the IP of the defender, and the type of attack executed. This will allow the trained model to recognize realistic attack patterns.

```
if attack == 1:
    attackerIP,defenderIP,attackname=random_attack(network)
    print(attackerIP + " attack: " + defenderIP + " :" + attackname)
else:
    attackerIP=0
    defenderIP=0
    attackname="no attack"
    print("no attack")
if userTraffic == 1:
    BRETAGNE.Generate_traffic.start(3)
time.sleep(5)
BRETAGNE.utils.tools.pcap_to_csv(pcap_file,csv_file)
time.sleep(1)
with open(csv_file, 'r', encoding='utf-8') as file:
    csv_string = file.read()
file_exists = os.path.isfile(dataset_file)

with open(dataset_file, mode='a', newline='') as file:
    writer = csv.writer(file)
    if not file_exists:
        writer.writerow(["Traffic", "Attack Flag", "Attacker IP", "Defender IP","Attack type"])
    writer.writerow([csv_string, attack, attackerIP, defenderIP, attackname])
```

The second functionality of the white agent is to evaluate a LLM. This functionality is similar to dataset generation. The program uses monitoring and selects a large number of random actions that will occur on the network. However, here, the traffic is directly sent to a LLM via BEDROCK, and the response is compared with what actually happened on the network. A scoring system, explained in the BRETAGNE research paper, is implemented. Moreover, the different elements are recorded in a CSV file for tracking the evaluation. This file contains an attack flag, the IP of the attacker, the IP of the defender, the LLM's response, its score, and an analysis of the LLM's response. This analysis is provided using the following logic:

```
respon = str(BRETAGNE.blueAgent.send_to_bedrock(csv_file,LLM)).lower()
if attack == 0 and "no" in respon:
    score=score+5
    analyse="OK +5"
elif attack == 1 and "no" in respon:
    score=score-1
    analyse="LLM missed an attack -1"
elif attack == 0 and re.search(regex,respon):
    score=score-5
    analyse="False positive -5"
elif attack == 1 and re.search(regex,respon) and not attackerIP in respon:
    score=score-3
    analyse="Bad IP attack -3"
elif attack == 1 and attackerIP in respon:
    score=score+5
    analyse="OK +5"
elif attack == 1 and "yes" and not re.search(regex,respon):
    score=score+1
    analyse="attack detected but no IP"
elif attack == 1 and not "yes" and not re.search(regex,respon):
    score=score+1
    analyse="attack detected but no IP"
elif attack == 0 and "yes" in respon:
    score=score-5
    analyse="False positive -5"
else:
    analyse="Out of context"
```

The evaluation of a LLM can be started with the command `--evaluate <LLM_name> <network_name>` and can be stopped with `ctrl+c`. The signal from this key combination is modified in the program to allow the script to run. The script `CountScore.py` will read the generated CSV file and calculate the statistics for each event:

```
with open(fichier_csv, mode='r', encoding='utf-8') as fichier:
    lecteur = csv.DictReader(fichier)
    total = 0
    OK = 0
    FalsePositive = 0
    Missedattack = 0
    BadAddress = 0
    OutOfContext = 0

    for ligne in lecteur:
        analyse = ligne.get('analyse')
        if analyse == 'OK +5':
            OK += 1
        elif analyse == 'False positive -5':
            FalsePositive += 1
        elif analyse == 'LLM missed an attack -1':
            Missedattack += 1
        elif analyse == 'Bad IP attack -3':
            BadAddress += 1
        elif analyse == 'Out of context':
            OutOfContext += 1
        total += 1
    print(f"{OK} OK on {total} requests: {OK/total*100:.2f}%")
    print(f"{FalsePositive} false positive on {total} requests: {FalsePositive/total*100:.2f}%")
    print(f"{Missedattack} undetected attacks on {total} requests: {Missedattack/total*100:.2f}%")
    print(f"{BadAddress} wrong address detected on {total} requests: {BadAddress/total*100:.2f}%")
    print(f"{OutOfContext}  Out of context on {total} requests: {OutOfContext/total*100:.2f}%")
```

This script can also be run separately with the command `python3` and the filename as a parameter to display the results from a previously generated file. C'est stastiques nous on permis d'etablir ce tableau concernant certains LLM :

LLM	CLAUDE 3.5 SONNET	MISTRAL LARGE 2	LLAMA 3.1 405B	GPT4o-mini	GEMINI 1.0 PRO
GOOD ANSWER	61,33%	57,65%	45,63%	47,18%	32,29%
FALSE POSITIVE	20,44%	25,88%	28,16%	40,14%	16,81%
UNDETECTED ATTACKS	16,57%	11,76%	9,71%	1,41%	36,13%
WRONG ADDRESS	1,66%	2,35%	6,80%	7,04%	7,56%
OUT OF CONTEXT	0%	2,36%	9,7%	4,23%	7,21%
OPEN SOURCE	✗	✓	✓	✗	✗
$ PER INPUT TOKENS $ PER OUTPUT TOKENS	3 per 1M 15 per 1M	3 per 1M 9 per 1M	5 per 1M 16 per 1M	0,15 per 1M 0,6 per 1M	0,5 per 1M 1,5 per 1M

This table shows that Claude 3.5 Sonnet provides the best results in this use case. Therefore, it is used by default by the blue agent. Additionally, it supports direct file processing, which allows for faster handling in the program compared to Mistral 2, for example.

4.3.4 Blue Agent

An autonomous blue agent using a LLM can also be deployed on a subnet. This agent will use the monitoring function to send network traffic to the LLM. The program will then process the LLM's response and automatically block the attacker using the SDN controller. The destination address `0.0.0.0/0` is used to block all connections.

```python
#This fonction check the response of the LLM in order to block the traffic
def check_response(rep):
    regex = r"\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b"
    if 'yes' in rep.lower():
        ip = re.search(regex,rep)
        if ip:
            ip = ip.group() + "/32"
            BRETAGNE.utils.SDN_action.BlockTraffic(ip,"0.0.0.0/0")

#Deploy a autonomous Blue Agent on the specified network
def run(network):
    pcap_file = f"simu/shared/capture/ovs_{network.lower()}.pcap"
    csv_file = f"simu/shared/capture/ovs_{network.lower()}.csv"
    while 1:
        BRETAGNE.utils.tools.clean_file(pcap_file,csv_file)
        BRETAGNE.monitoring.monitor(network.lower())
        time.sleep(15)
        BRETAGNE.utils.tools.pcap_to_csv(pcap_file,csv_file)
        time.sleep(1)
        respon = send_to_bedrock(csv_file, "sonnet")
        check_response(respon)
        print(respon)
```

For example, you can deploy a blue agent with the command `--BlueAgent <network_name>` and then manually launch an attack on the network to observe the creation of blocking rules on the web interface of the SDN controller.

5 Example of a dataset generated by the white agent

Traffic	Attack Flag	Attacker IP	Defender IP	Attack type	Rec
1 0.000000 7a:56:9d:26:93:c8 → ARP 66 Who has 1.1.2.25…	0	0	0	no attack	1
1 0.000000 86:4e:ca:96:a2:08 → ARP 66 Who has 100.100…	1	1.1.1.97	1.2.2.6	brute force ssh	1
1 0.000000 96:12:bb:f5:3f:a4 → ARP 66 Who has 1.1.1.4? …	1	1.1.1.216	1.1.1.4	brute force ssh	1
	0	0	0	no attack	1
1 0.000000 76:1b:86:0c:8e:9a → ARP 66 Who has 1.1.2.4?…	1	1.1.1.42	1.1.2.6	web ddos	1
1 0.000000 8e:e0:a1:50:39:4a → ARP 66 Who has 192.16…	0	0	0	no attack	1
1 0.000000 ca:ba:dc:c3:b1:6f → ARP 66 Who has 100.100.…	0	0	0	no attack	1
	0	0	0	no attack	1
	0	0	0	no attack	1
1 0.000000 26:4e:1a:18:c5:53 → ARP 66 Who has 192.168…	1	1.1.1.38	1.2.1.10	portscan	1
1 0.000000 56:18:31:23:dd:c3 → ARP 66 Who has 192.168…	0	0	0	no attack	1
1 0.000000 76:1b:86:0c:8e:9a → ARP 66 Who has 1.1.2.25…	1	1.1.1.91	1.1.2.7	web ddos	1
1 0.000000 56:c1:cb:04:d7:80 → ARP 66 Who has 1.2.2.25…	0	0	0	no attack	1
	1	1.1.1.156	100.100.2.7	web ddos	1
1 0.000000 56:5a:0d:74:ab:64 → ARP 66 Who has 192.16…	1	1.1.1.45	1.1.2.5	web ddos	1
	0	0	0	no attack	1
	1	1.1.1.163	100.100.0.6	web ddos	1
	0	0	0	no attack	1
1 0.000000 32:c9:24:91:b4:45 → ARP 66 Who has 1.1.1.25…	1	1.1.1.150	1.1.2.4	portscan	1

6 Example of an LLM evaluation file

Attack Flag	Attacker IP	Attack type	LLM response	score	analyse
0	NA	no attack	yes, 1.1.2.10 and 100.100.1.8	95	False positive -5
0	NA	no attack	no attack detected. the traffic appears to be normal arp req…	100	OK +5
1	1.1.1.57	portscan	yes, 1.1.1.57	105	OK +5
0	NA	no attack	no, there is no clear evidence of an attack in this network tr…	110	OK +5
0	NA	no attack	yes, 100.100.0.12	105	False positive -5
1	1.1.1.20	portscan	yes, an attack has taken place. the attacker ip is 1.1.1.20.	110	OK +5
1	1.1.1.76	brute force ssh	yes, 1.1.1.76 (attacker ip)	115	OK +5
0	NA	no attack	yes, potential arp flooding attack from multiple ips: 100.100…	110	False positive -5
1	1.1.1.37	portscan	yes, 1.1.1.37 the traffic shows repeated arp requests from t…	115	OK +5
1	1.1.1.157	web ddos	based on the information provided ("no traffic"), there is not…	114	LLM missed an attack -1
1	1.1.1.70	web ddos	based on the information provided ("no traffic"), there is not…	113	LLM missed an attack -1
0	NA	no attack	yes, 1.1.1.4	108	False positive -5
0	NA	no attack	yes, 192.168.1.4	103	False positive -5
0	NA	no attack	yes, an attack is likely taking place. the attacker ip is 100.1…	98	False positive -5
0	NA	no attack	based on the information provided ("no traffic"), there is not…	103	OK +5
0	NA	no attack	without any actual network traffic data provided, it's not pos…	108	OK +5
0	NA	no attack	yes, potential arp flooding attack from multiple ips: 1.2.1.7 …	103	False positive -5
0	NA	no attack	no, there is no clear evidence of an attack in this network tr…	108	OK +5
1	1.1.1.128	brute force ssh	yes, 1.1.1.128	113	OK +5