AIC-5101C

AWS AND CLOUD SECURITY SERVICES

PROJECT – VPC AND IDS

A Project Report

Submitted by

Antoine DEBAUGE

Artur KAZARIAN

Shakthivel MURUGAVEL

Victorgabrielmendes SUNDERMANN

of

E5 - ARTIFICIAL INTELLIGENCE AND CYBERSECURITY

Under the Guidance of

Monsieur Mawloud OMAR

93160, Noisy-le-Grand, France

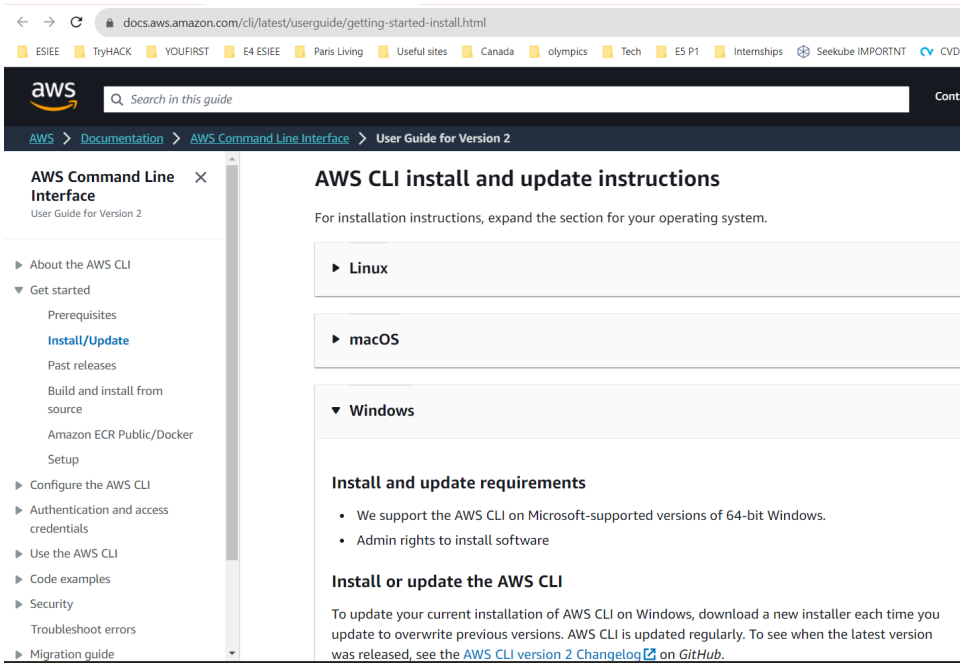
09th November 2023

ESIEE PARIS

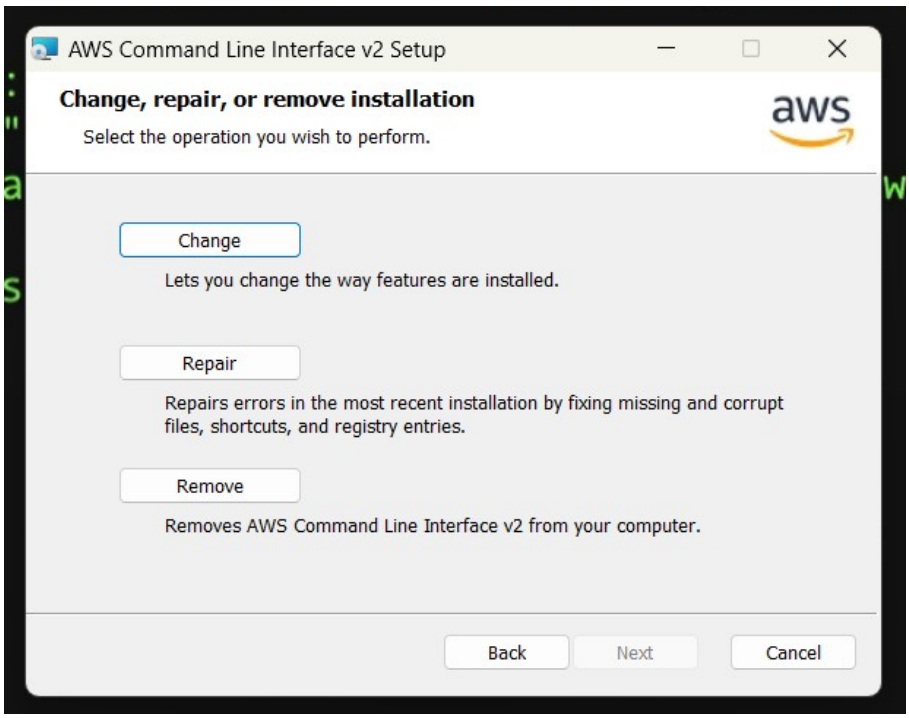
TABLE OF CONTENTS

SETTING UP CLI

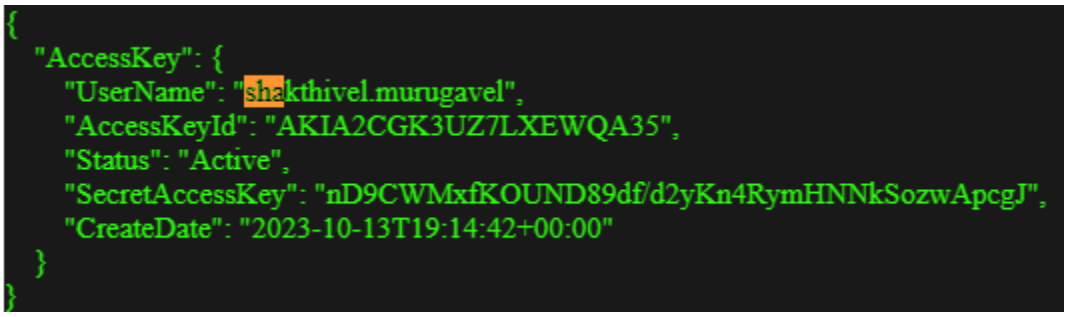
The objective of this Lab is to practice AWS IAM using CLI. The AWS CLI is set up following the instructions from the Amazon's official website.



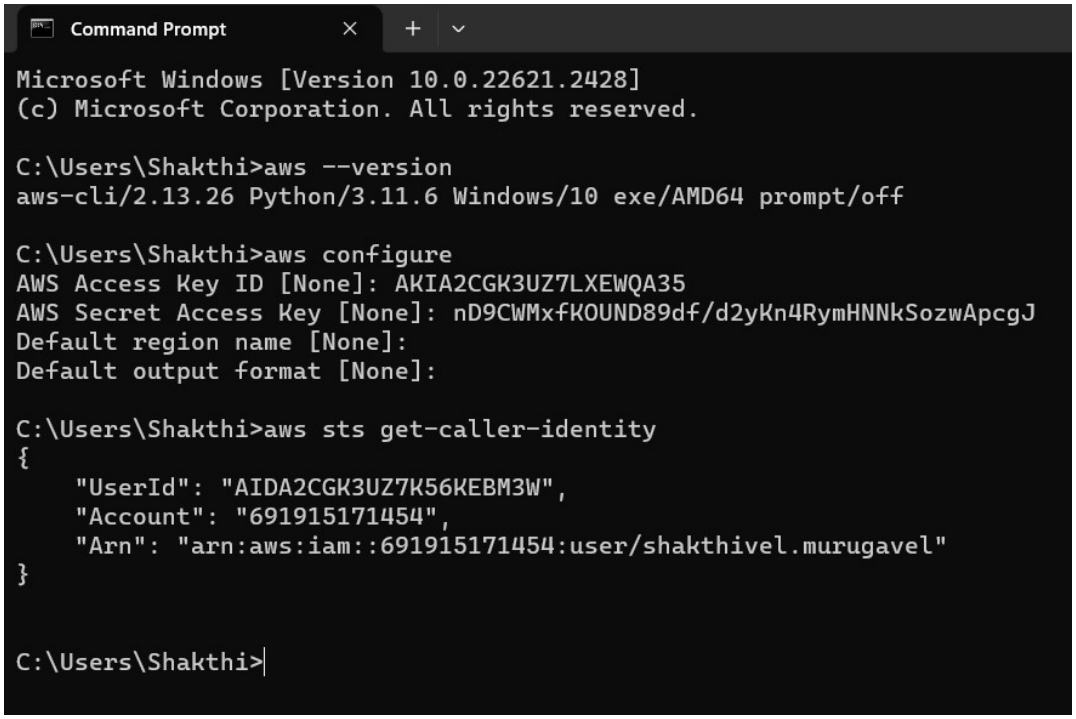
A package installer is used to setup the CLI and the setup is completed successfully.



The CLI is configured using the command ***aws configure*** with the credentials given in the instructions.



```
{
  "AccessKey": {
    "UserName": "shakthivel.murugavel",
    "AccessKeyId": "AKIA2CGK3UZ7LXEWQA35",
    "Status": "Active",
    "SecretAccessKey": "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",
    "CreateDate": "2023-10-13T19:14:42+00:00"
  }
}
```



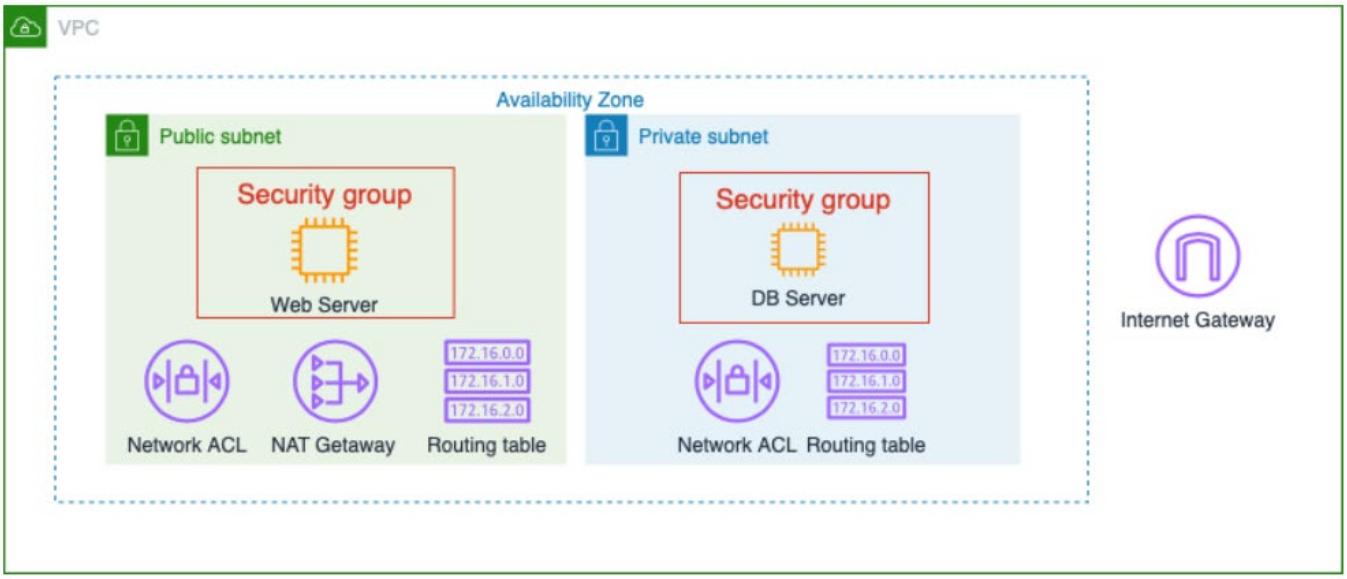
```
Command Prompt
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Shakthi>aws --version
aws-cli/2.13.26 Python/3.11.6 Windows/10 exe/AMD64 prompt/off

C:\Users\Shakthi>aws configure
AWS Access Key ID [None]: AKIA2CGK3UZ7LXEWQA35
AWS Secret Access Key [None]: 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
Default region name [None]:
Default output format [None]:

C:\Users\Shakthi>aws sts get-caller-identity
{
    "UserId": "AIDA2CGK3UZ7K56KEBM3W",
    "Account": "691915171454",
    "Arn": "arn:aws:iam::691915171454:user/shakthivel.murugavel"
}


C:\Users\Shakthi>
```

PART 1

MOUNT VPC

Write an .sh CLI script that mounts the VPC illustrated in the figure below.

REQUIRED ARCHITECTURE



BASH SCRIPT

C: > Users > Shakthi > Desktop > AWS > $ trial.sh

```bash
#!/bin/bash

# VPC configuration
AWS_REGION="us-east-1"
VPC_IP="10.0.0.0/16"
PUB_SUB_ID="10.0.1.0/24"
PVT_SUB_ID="10.0.2.0/24"

# Create the VPC
response=$(aws ec2 create-vpc --cidr-block $VPC_IP --region $AWS_REGION)
# Extract VPC ID using jq and store it in a variable
vpc_id=$(echo $response | jq -r '.Vpc.VpcId')

aws ec2 create-tags --resources $vpc_id --tags Key=Name,Value=MyVPC --region $AWS_REGION
# Display the VPC ID
echo "VPC ID: $vpc_id"


# Create the Public Subnet
response1=$(aws ec2 create-subnet --vpc-id $vpc_id --cidr-block $PUB_SUB_ID --region $AWS_REGION)
public_subnet_id=$(echo $response1 | jq -r '.Subnet.SubnetId')

aws ec2 create-tags --resources $public_subnet_id --tags Key=Name,Value=myPublicSubnet --region $AWS_REGION
echo "Public Subnet ID: $public_subnet_id"


# Create the Private Subnet
response3=$(aws ec2 create-subnet --vpc-id $vpc_id --cidr-block $PVT_SUB_ID --region $AWS_REGION)
private_subnet_id=$(echo $response3 | jq -r '.Subnet.SubnetId')

aws ec2 create-tags --resources $private_subnet_id --tags Key=Name,Value=myPrivateSubnet --region $AWS_REGION
echo "Private Subnet ID: $private_subnet_id"
```

EXPLANATION

1. Create VPC: `aws ec2 create-vpc`
   This command initiates the creation of the VPC.

2. Create Subnets:
Two subnets are created: one for public resources and another for private resources.
`aws ec2 create-subnet --vpc-id $vpc_id --cidr-block $PUB_SUB_ID`
`aws ec2 create-subnet --vpc-id $vpc_id --cidr-block $PVT_SUB_ID`
Each subnet serves a different purpose - public for resources that need direct internet access and private for those that do not.

3. Create Internet Gateway (IGW):
`aws ec2 create-internet-gateway` creates an internet gateway that enables communication between the VPC and the internet.
`aws ec2 attach-internet-gateway --internet-gateway-id $internet_gateway_id` attaches the internet gateway to the VPC.

4. Create NAT Gateway:
`aws ec2 allocate-address --domain vpc --region` allocates an Elastic IP address used for the NAT gateway.
`aws ec2 create-nat-gateway --subnet-id $public_subnet_id` creates a NAT gateway in the public subnet to allow private resources to access the internet while remaining private.

5. Route Tables:
Two route tables are created, one for each subnet.
`aws ec2 create-route-table --vpc-id $vpc_id` creates the route tables.
Routes are added to each route table using `aws ec2 create-route`.
Public subnet route table uses the internet gateway, and the private subnet route table uses the NAT gateway.

6. Associate Route Tables:
`aws ec2 associate-route-table --route-table-id $route_table_id --subnet-id $public_subnet_id`
`aws ec2 associate-route-table --route-table-id $route_table_id2 --subnet-id $private_subnet_id`
Associates the created route tables with their respective subnets.

7. Security Groups:

Two security groups are created: one for web servers and one for a database server.

`aws ec2 create-security-group` commands are used to create them.

`aws ec2 authorize-security-group-ingress` rules are added to allow inbound traffic for SSH (port 22), HTTP (port 80), etc.

8. Key Pair:

`aws ec2 create-key-pair` generates a key pair for SSH access to instances within the VPC.

VPC ARCHITECTURE:

The architecture created involves a VPC with:

Public and private subnets for different types of resources.

Internet Gateway for the public subnet to allow internet access.

NAT Gateway for the private subnet to provide internet access to private resources while maintaining security.

Route tables configured to direct traffic based on whether it's destined for the internet or internal network.

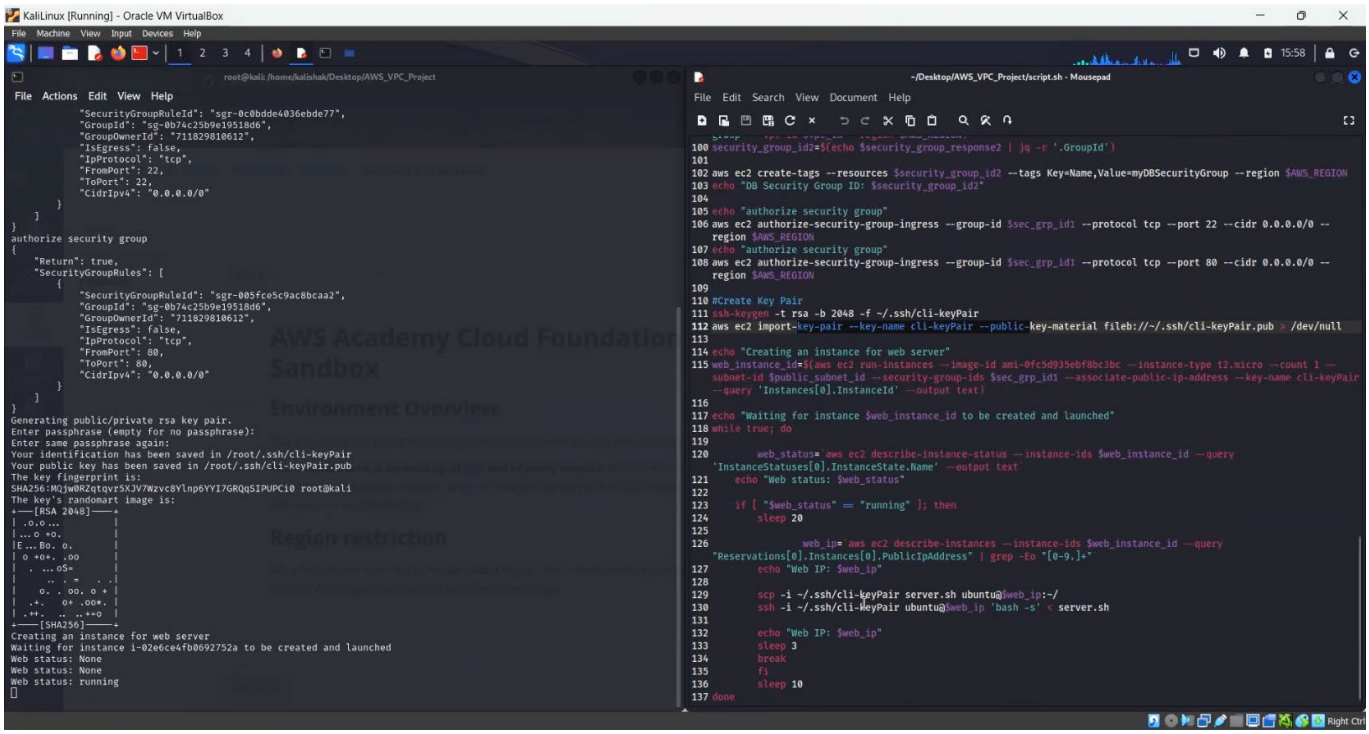
Security groups to control inbound and outbound traffic to instances. Key Pair generated for secure access to instances via SSH.

This setup segregates resources and controls their access and communication both within the VPC and with external networks.

OUTPUT

```
MINGW64:/c/Users/Shakthi/Desktop/AWS

mail2@Shak-Alien MINGW64 ~/Desktop/AWS
$ ./trial.sh
VPC ID: vpc-0b8810b584c8ace19
Public Subnet ID: subnet-040131ae7973c4c34
Private Subnet ID: subnet-0143588d19c1bb53b
InternetGateway ID: igw-0811d41c0f5949c2f
Internet Gateway Attached to VPC
Public Route Table ID: rtb-02b4b68bc015926fb
Private Route Table ID: rtb-0be3adac6efe2078b
Create route:
{
    "Return": true
}

{
    "Return": true
}

Route table association public ID: rtbassoc-06ae46b12f1c8791b
Route table association private ID: rtbassoc-0dcc6499a5303a631
Web Security Group ID: sg-0a65af818e27a3cd5
DB Security Group ID: sg-010d8808a149db6ed
authorize security group
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-03063857db0a74bcd",
            "GroupId": "sg-0a65af818e27a3cd5",
            "GroupOwnerId": "711829810612",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 22,
            "ToPort": 22,
            "CidrIpv4": "0.0.0.0/0"
        }
    ]
}

authorize security group
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-0c20ad650c8cd39c6",
            "GroupId": "sg-0a65af818e27a3cd5",
            "GroupOwnerId": "711829810612",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 80,
            "ToPort": 80,
            "CidrIpv4": "0.0.0.0/0"
        }
    ]
}
```

```
mail2@Shak-Alien MINGW64 ~/Desktop/AWS
$ aws ec2 create-key-pair --key-name cli-keyPair --query 'KeyMaterial' --output text > cli-keyPair.pem --region "us-east-1"

mail2@Shak-Alien MINGW64 ~/Desktop/AWS
$ chmod 400 cli-keyPair.pem

mail2@Shak-Alien MINGW64 ~/Desktop/AWS
$
```



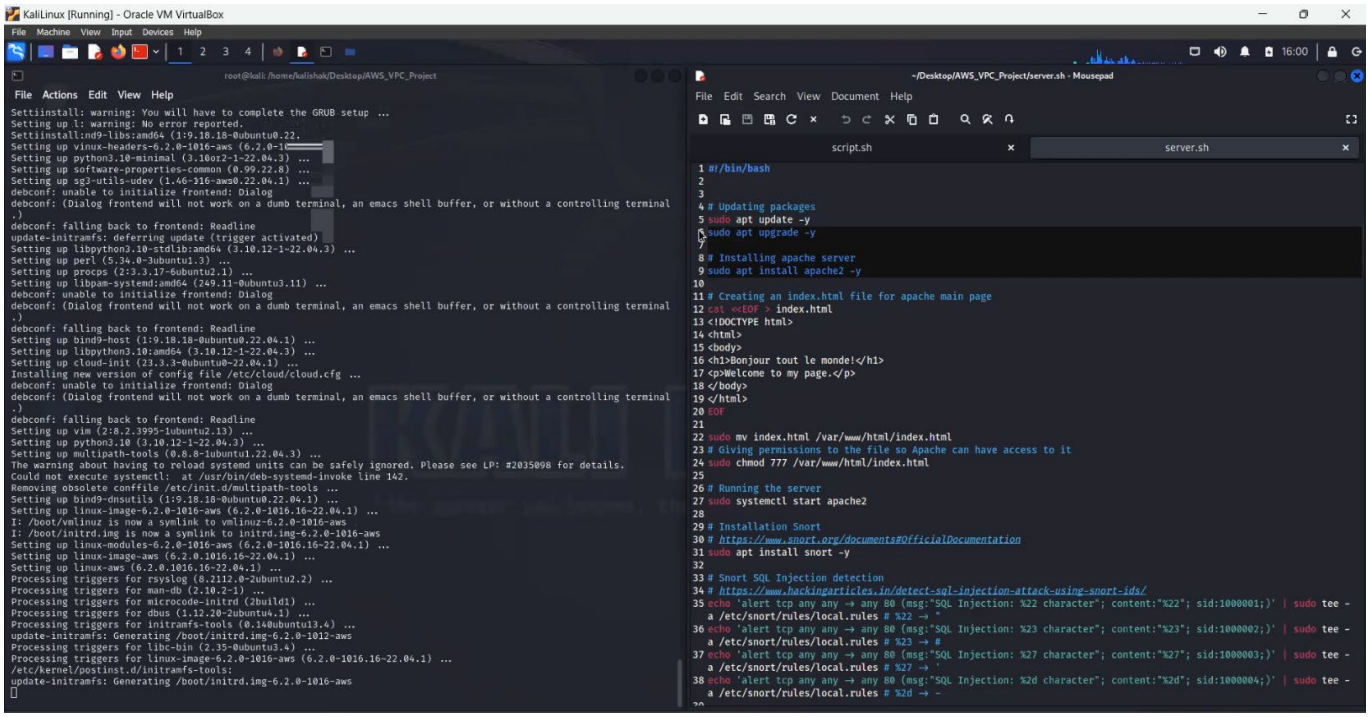
1.2 DEPLOY AND CONFIGURE SERVER

Deploy and configure the servers (e.g., Apache, MariaDB) so that they will be fully operational.

CREATING AN EC2 INSTANCE

To deploy the apache server we have to create an EC2 instance first based on the previously created VPC.

```
echo "Creating an instance for web server"
web_instance_id=$(aws ec2 run-instances --image-id ami-0533f2ba8a1995cf9 --instance-type t2.micro --count 1 --subnet-id $public_sub

echo "Waiting for instance $web_instance_id to be created and launched"

sleep 200

web_public_dns=$(aws ec2 describe-instances --instance-ids=$web_instance_id --query 'Reservations[].Instances[].PublicDnsName')
echo $web_public_dns
web_instance_dns=$(echo $web_public_dns | tr -d '[" ]')

echo "Web instance dns: $web_instance_dns"

ssh -i "cli-keyPair.pem" ec2-user@$web_instance_dns 'bash -s' < install_apache.sh
```



The script above describes the following steps:

- Create an instance, save the instance id into a variable;
- Wait for 200 seconds for the instance to launch (normally it is enough);
- Extract the public DNS address from the instance by providing it's id;
- Connect to the instance by SSH.

After the connection is established, the install_apache.sh script is executed on the instance.

Installing Apache on the instance

The code for installing and launching an Apache web-server can be found at install_apache.sh script. Let's have a look at it:

```
#!/bin/sh

# Updating packages
sudo yum update -y

# Installing apache server
sudo yum install -y httpd

# Creating an index.html file for apache main page
{
echo "<!DOCTYPE html>"
echo "<html>"
echo "<body>"
echo "<h1>Bonjour tout le monde!</h1>"
echo "<p>Welcome to my page.</p>"
echo "</body>"
echo "</html>"
}>> index.html

sudo mv index.html /var/www/html/index.html
# Giving permissions to the file so Apache can have access to it
sudo chmod 777 /var/www/html/index.html

# Running the server
sudo service httpd start
```

The script describes the following steps:

- Updating the existing packages on the instance;
- Installing Apache web-server;
- Creating an index.html file as a default page;
- Moving the file to a home directory of an Apache;
- Giving the permissions so web-app can read the file;
- Launching the web-server.

After the installation is complete, the web-page can be accessed by the public URL:

PART 2

NETWORK TRAFFIC ANALYSIS

Attempt 1:

In order to create a network traffic analysis we will set up a web application and a Invasion Detection System (IDS) to protect our service from attacks and undesired usage.

The web application of choice is Damn Vulnerable Web Application (DVWA), this application already has features to test cybersecurity attacks, such as SQL injection. After creating the IDS ec2 instance we download and configure the application with the following script.

```
$ WebAppSQLinj.sh
1   #curl https://github.com/digininja/DVWA/archive/master.zip
2   git clone https://github.com/digininja/DVWA
3
4   chmod -R 777 DVWA/
5
```

DVWA Installation Script

Now that the application is functional, we need to install snort, configure it and create the rules that will detect any attempt of SQL injection. We have set up 5 rules to detect an injection attack, the first two detect attacks that exploit apostrophes and quotation markers, while the rest tackle Inline Comments, Boolean-based injection, UNION keyword and Manual injection respectively.

These steps are done with the following script:

```
$ NTA.sh
1   #Install IDS directly to the web server
2   #Detect SQL injection attacks, without traffic mirroring
3
4   # Installation Snort
5   # https://www.snort.org/documents#OfficialDocumentation
6   sudo apt install snort
7
8   # Snort IDS mode
9   # Possible to change config in cmd line with --lua
10  snort -c /etc/snort/snort.lua -r /var/www/dvwa
11
12  # Snort SQL Injection detection
13  # https://www.hackingarticles.in/detect-sql-injection-attack-using-snort-ids/
14  alert tcp any any -> any 80 (msg: "Error Based SQL Injection Detected"; content: "%27" ; sid:100000011; )
15  alert tcp any any -> any 80 (msg: "Error Based SQL Injection Detected"; content: "%22" ; sid:100000012; )
16
17  # https://medium.com/@johnsamuelthiongo52/sql-injection-ids-using-snort-ffd639cb0f3f
18  alert tcp any any -> any any (msg:"Possible SQL Injection – Inline Comments Detected"; flow:to_server,established; content:"GET";
19  alert tcp any any -> any any (msg:"Possible Boolean-based Blind SQL Injection Attempt"; flow:to_server,established; content:"GET"
20  alert tcp any any -> any 80 (msg:"Possible SQL Injection – UNION keyword detected"; flow:to_server,established; content:"UNION";
21  alert tcp any any -> any 80 (msg:"Possible Manual Injection detected"; flow:to_server,established; content:"GET"; http_method; co
```

Snort installation and rules

With the application running and the rules set, we can test whether our setup works or not, the command below is an example of SQL injection that can be passed to the website:

http://10.0.1.0/dvwa/vulnerabilities/sqli/?id='%'&Submit=Submit#

2.1 EC2 INSRANCE AND SQL INJECTION

Attempt 2: (Success!)

We tried with DVWA but faced issues. Since, our own websites is vulnerable too, we decided to continue the SQLi attacks on the website we hosted. We installed the server on to the public subnet and hosted our own vulnerable website to perform the attacks.

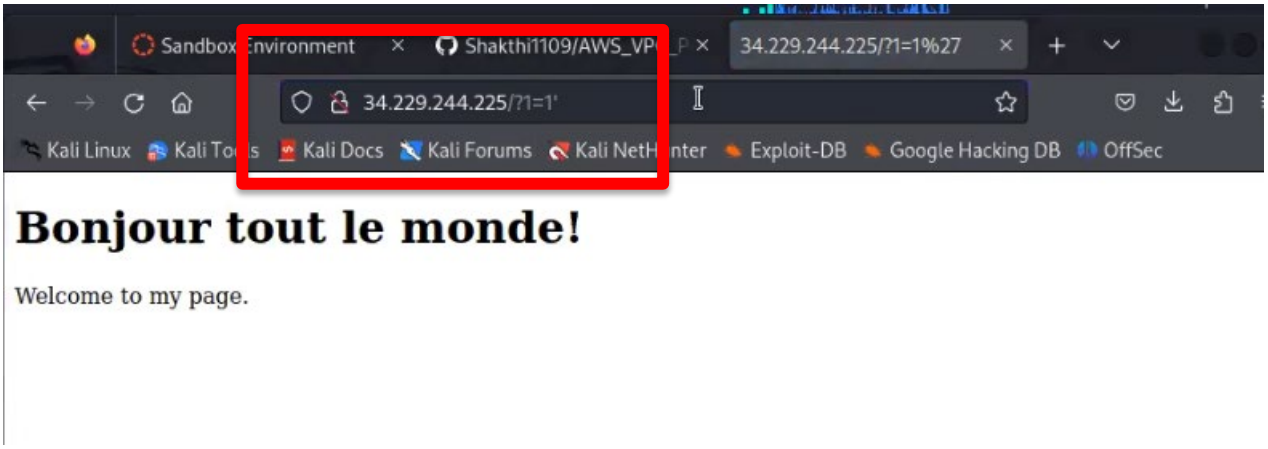
Below are the commands used:

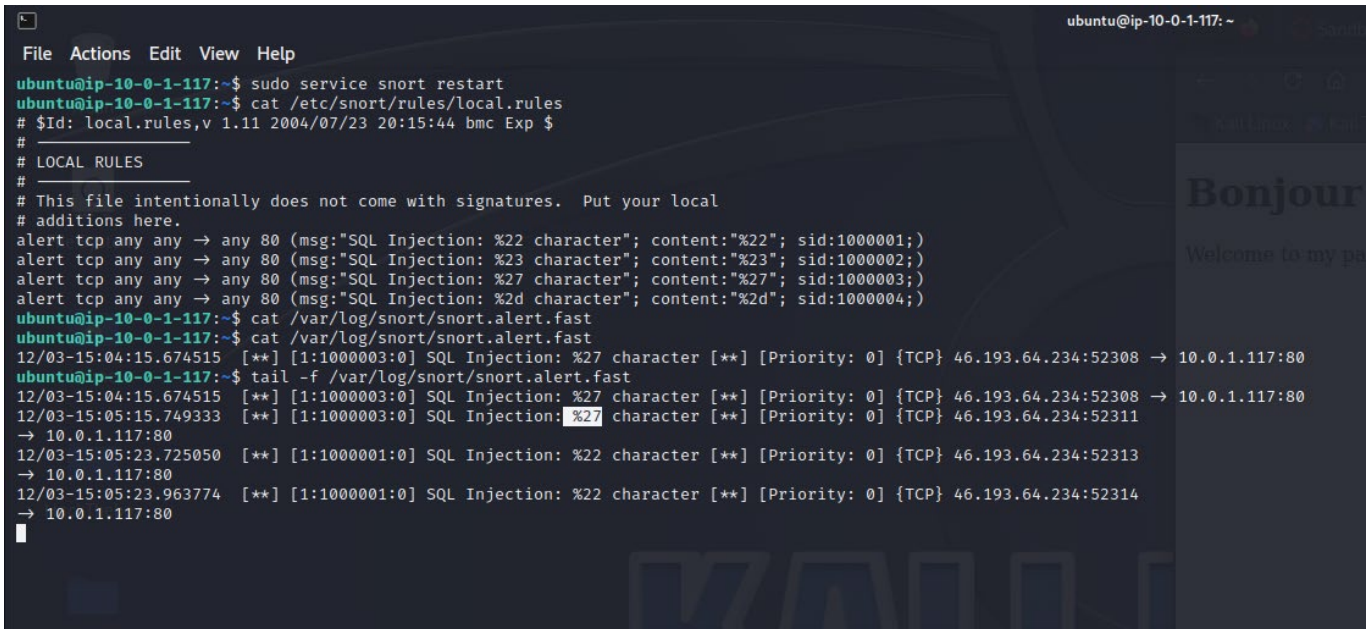
```
1 # ssh connection into Web Server
2 ssh -i ~/.ssh/cli-keyPair ubuntu@<WEB_IP>
3
4 # check snort logs
5 cat /var/log/snort/snort.alert.fast
6 tail -f /var/log/snort/snort.alert.fast
7
8 ##################
9 ##### DEBUG #####
10 ##################
11
12 # check snort service
13 sudo service snort start/stop/restart/status
14
15 # check local rules
16 cat /etc/snort/rules/local.rules
17
18 # check process running
19 ps aux | grep -i "snort"
20
21 # kill running process
22 sudo kill <ID_PROCESS>
23
```

Below are the rules we defined for SQLi attacks:

```
ubuntu@ip-10-0-1-117: ~
File  Actions  Edit  View  Help
ubuntu@ip-10-0-1-117:~$ sudo service snort restart
ubuntu@ip-10-0-1-117:~$ cat /etc/snort/rules/local.rules
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ----------------
# LOCAL RULES
# ----------------
# This file intentionally does not come with signatures.  Put your local
# additions here.
alert tcp any any -> any 80 (msg:"SQL Injection: %22 character"; content:"%22"; sid:1000001;)
alert tcp any any -> any 80 (msg:"SQL Injection: %23 character"; content:"%23"; sid:1000002;)
alert tcp any any -> any 80 (msg:"SQL Injection: %27 character"; content:"%27"; sid:1000003;)
alert tcp any any -> any 80 (msg:"SQL Injection: %2d character"; content:"%2d"; sid:1000004;)
```

In our hosted website, when an attack is attempted as shown below, the scrip captures it and hence the SQL injections are detected.





The above screenshot shows the bash script detecting SQLi attacks using snort.

ESIEE PARIS

PART 3

CONCLUSION

We successfully Setup VPC as per the instructions given (Part 1) and performed SQLi attacks on a vulnerable site and detected the attacks using snort with a bash script (Part 2). Hence, we successfully complete part 1 and part 2 of the AWS Mini Project.

GitHub: https://github.com/Shakthi1109/AWS_VPC_Project

YouTube: https://youtu.be/oqNhdRzBkDs

THANK YOU