**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**

**Logo

Description automatically generated**

93160, Noisy-le-Grand, France

**09th November 2023**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **TITLE** | **PAGE NO.** |
| **1. VPC** | **05** |
| **1.1 MOUNT VPC** | **05** |
| **1.2 DEPLOY AND CONFIGURE SERVER** | **09** |
| **2. NETWORK TRAFFIC ANALYSIS** | **11** |
| **2.1 EC2 INSRANCE AND SQL INJECTION** | **11** |
| **2.2 MIRRORING SERVICE** | **12** |
| **3. GITHUB** | **12** |

**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 screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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

A green text on a black background

Description automatically generated

A screenshot of a computer

Description automatically generated

**PART 1**

**MOUNT VPC**

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

**REQUIRED ARCHITECTURE**

A screenshot of a computer program

Description automatically generated

**BASH SCRIPT**

A screenshot of a computer program

Description automatically generated

**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**

A computer screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

**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.

A computer screen with text

Description automatically generated

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:

A screenshot of a computer program

Description automatically generated

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:

A screenshot of a computer

Description automatically generated

**PART 2**

**NETWORK TRAFFIC ANALYSIS**

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.

A screenshot of a computer application

Description automatically generated

**DVWA home page**

A computer screen with a black background

Description automatically generated

**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:

A screenshot of a computer program

Description automatically generated

**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#

The efficacy of our implementation was unable to be tested due to a few technical difficulties. We

aimed to automate the injection process on the Shell script but fell short in finding a way to open the application and do it, instead a python script would be better for this task, but it ultimately falls out of the scope of the project.

**PART 3**

**GITHUB PROJECT REPOSITORY**

<https://github.com/Shakthi1109/AWS_VPC_Project>