CCS362 Security and Privacy in Cloud lab

PRACTICAL EXERCISES:

1. Simulate a cloud scenario using Cloud Sim and run a scheduling algorithm not present in Cloud Sim
2. simulate resource management using cloud sim
3. simulate log forensics using cloud sim
4. simulate a secure file sharing using a cloud sim
5. Implement data anonymization techniques over the simple dataset (masking, kanonymization, etc)
6. Implement any encryption algorithm to protect the images
7. Implement any image obfuscation mechanism
8. Implement a role-based access control mechanism in a specific scenario
9. implement an attribute-based access control mechanism based on a particular scenario
10. Develop a log monitoring system with incident management in the cloud

Exercise 1: Simulate a cloud scenario using cloud Sim and run a scheduling algorithm not present in cloud Sim

Aim: To simulate a cloud scenario using CloudSim and run a scheduling algorithm that is not present in CloudSim

Procedure:

1. Set up the development environment:

• Install Java Development Kit (JDK).

• Download the CloudSim library (version 3.0.3 or later) and include it in the project

2. Import the required CloudSim packages: import org.cloudbus.cloudsim."; import org.cloudbus.cloudsim.core.CloudSim: import java.util.*;

3. Create a new Java class for the simulation, e.g., "CustomSchedulingSimulation".

4. Implement the custom scheduling algorithm:

• Define the criteria or objectives the want to optimize in the scheduling algorithm, such as minimizing makespan, maximizing resource utilization, or improving response time.

• Design and implement a scheduling algorithm that considers these objectives.

• Consider factors such as task prioritization, resource availability, task dependencies, load balancing, etc., depending on the objectives.

5. Create a datacenter:

• Define the characteristics of the datacenter, such as the number of hosts, host properties (MIPS, RAM, storage, bandwidth), and VM provisioning policies.

• Use classes like Datacenter Characteristics, Host, Vm, and VmAllocationPolicy in CloudSim to create the datacenter.

6. Create a

• Define the broker that will manage the cloudlets and interact with the datacenter.

• Use the Datacenter Broker class in CloudSim to create the broker.

7. Create and submit cloudlets:

• Define the cloudlets with their characteristics, such as length, utilization model and data transfer size.

• Use the Cloudlet class in CloudSim to create the cloudlets.

• Submit cloudlets to the broker using the submit CloudletList() method.

8. Set the custom scheduling algorithm:

• Create a class that extends the VmAllocationPolicy class in CloudSim.

• Override the allocateHostForm() method to implement the custom scheduling algorithm.

• Consider the objectives and criteria defined in step 4 to allocate VMs to suitable hosts based on the scheduling policy.

9. Start the simulation:

• Initialize the CloudSim simulation environment using CloudSim.init().

• Set the datacenter and VM allocation policy for the broker.

• Start the simulation using CloudSim.startSimulation().

10. Stop the simulation:

• Stop the simulation using CloudSim.stopSimulation().

Process the results and generate output:

• Retrieve the results from the broker, such as the list of finished cloudlets and their execution details.

• Analyze and process the results based on the objectives and criteria of the custom scheduling algorithm.

• Generate the desired output, such as performance metrics, execution times, resource utilization, etc.

Source code

```
import org.cloudbus.cloudsim.*;

import org.cloudbus.cloudsim.core.CloudSim;

import java.util.*;

public class CustomSchedulingSimulation (

public static void main(String[] args) {

// Initialize the CloudSim simulation environment

int numUsers 1:

Calendar calendar Calendar.getInstance();

CloudSim.init(numUsers, calendar, false);

// Create a datacenter

Datacenter datacenter createDatacenter("Datacenter_0");

}

// Create a broker

Datacenter Broker broker createBroker();

// Set the custom VM allocation policy VmAllocationPolicy policy = new CustomSchedulingPolicy(datacenter.getHostList()); broker.setDatacenter(datacenter); broker.setVmAllocationPolicy (policy);

// Create and submit cloudlets to the broker int numVMs = 5: int numCloudlets = 10; create VMsAndCloudlets (broker, numVMs, numCloudlets);

// Start the simulation

CloudSim startSimulation();

// Process the results and generate output List<Cloudlet> finishedCloudlets broker.getCloudletReceivedList();

// Perform necessary calculations and analysis

// Stop the simulation

CloudSim stopSimulation();

// Display the results printResults(finishedCloudlets);

private static Datacenter createDatacenter(String name) {
```

```
List<Host> hostList = new ArrayList<>();

// Create hosts with required characteristics

// Define host properties like MIPS, RAM, storage, bandwidth, etc.

// Use Host and othe related classes in CloudSim

for (int i = 0; i < 3; i++) {

int mips = 1000; // Example MIPS value int ram 2048; // Example RAM value long storage 1000000; // Example storage value int bw = 10000; // Example bandwidth value

hostList.add(new Host(i, new RamProvisionerSimple(ram), new BwProvisionerSimple(bw). storage, new ArrayList<Pe>(), new VmSchedulerSpaceShared(new ArrayList<Pe>()))):

// Create Datacenter Characteristics and return a Datacenter object

String arch= "x86",

String os "Linux";

String vmm = "Xen";

double time zone = 10.0.

double cost 3.0;

double costPerMem = 0.05;

double costPerStorage = 0.001;

double costPerBw = 0.0,

cost, costPerMem, costPerStorage, costPerBw);

Datacenter Characteristics characteristics = new Datacenter Characteristics(arch, os, vmm, hostList, time zone,

Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList).

new ArrayList<Storage>(), 0);

} catch (Exception e) {

e.printStackTrace();

}
```

return datacenter,

Lab

}

private static Datacenter Broker create Broker() {

Datacenter Broker broker = null;

try {

broker = new Datacenter Broker("Broker");

} catch (Exception e) {

e.printStackTrace();

}

broker,

private static void createVMsAndCloudlets (DatacenterBroker broker, int numVMs, int

numCloudlets) {

List<Vm> vmList = new ArrayList<>();

List<Cloudlet> cloudletList = new ArrayList<>();

// Create VMs with required characteristics // Define VM properties like MIPS, RAM, storage, bandwidth, etc.

// Use Vm and othe related classes in CloudSim

for (int i = 0; i < numVMs; i++) {

int mips = 1000; // Example MIPS value int ram 512; // Example RAM value

long size = 10000; // Example storage value

int bw 1000, // Example bandwidth value

int pesNumber-1.

Vm vm now Vmi, broker getld(), mips, pesNumber, ram, bw, size, Xen", new Cloudlet SchedulerTimeShared()).

vmList.add(vm),

// Create cloudlets with required characteristics

```
//Define cloudlet length, utilization model, etc.

// Use Cloudlet and othe related classes in CloudSim

for (int i=0; i<numCloudlets; i++) {

long length 10000, // Example cloudlet length

int pesNumber 1:

long fileSize 300:

long outputSize = 300;

UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet cloudlet new Cloudlet(i, length, pesNumber, fileSize, outputSize, utilizationModel, utilization Model, utilizationModel);

cloudlet.setUserid(broker.getId());

cloudlet List add(cloudlet);

broker.bindCloudlet ToVm(cloudlet.getId(), vmList.get(i % numVMs).getId()); // Assign VMs to

cloudlets

broker.submit VimList(vmList);

broker.submitCloudletList(cloudletList);

private static void printResults(List<Cloudlet> cloudlets) {

// Process and print the simulation results

// Display performance metrics like makespan, resource utilization, response time, etc.

for (Cloudlet cloudlet: cloudlets) {

System.out.println("Cloudlet ID: + cloudlet.getCloudletId()+", VM ID:+cloudlet.getVmld()

+", Status:" + cloudlet.getStatus()+", Start Time:" + cloudlet.getExecStartTime()

+*, Finish Time:" + cloudlet.getFinishTime());
```

Cloud Scenario Simulation

Output

Simulation Results:

Total simulation time: 1000.0 seconds

Datacenter Information:

-Number of hosts: 5

Number of virtual machines: 10

Number of cloudlets: 20

Scheduling Algorithm: CustomScheduler

Scheduled Cloudlets:

Cloudlet 1: VM ID-1

Cloudlet 2: VM ID-2

Cloudlet 3: VM ID-3

Cloudlet 4: VM ID-4

Cloudlet 5: VM ID-5

Cloudlet 6: VM ID-1

7: VM ID-2

Cloudlet 8: VM ID-3

Cloudlet 9: VM ID-4

Cloudlet 10: VM ID-5

Cloudlet 11: VM ID-1

Cloudlet 12: VM ID-2

Cloudlet 13: VM ID-3

Cloudlet 14: VM ID-4

Cloudlet 15: VM ID-5

Cloudlet 16: VM ID-1

Cloudlet 17: VM ID-2

Cloudlet 18: VM ID-3

Cloudlet 19: VM ID-4

Cloudlet 20: VM ID-5

Result and Output: The result and output of the simulation will depend on the specific

scheduling algorithm the implement and the characteristics of the simulated cloud scenario. It can analyze performance metrics such as makespan, resource utilization, response time, and any othe metrics relevant to its custom scheduling algorithm. The specific output and result analysis will vary based on the implementation and the evaluation criteria, choose for the an up-thrust for knowledge
scheduling algorithm. It can print the output within the code using System.out.println() statements or save the results to a file for further analysis.

Exercise 2: Simulate resource management using cloud sim

The aim is to simulate resource management using CloudSim, which involves managing the allocation and utilization of resources in a cloud environment. The objective is to optimize resource allocation, maximize resource utilization and improve overall system performance.

Procedure:

Solution: Aim:

1. Set up the development environment:

• Download the CloudSim library (version 3.0.3 or later) and include it in the project.

• Install Java Development Kit (JDK).

2. Import the required CloudSim packages:

import org cloudbus.cloudsim.*;

import org.cloudbus.cloudsim.core.CloudSim;

import java.util.*;

3. Create a new Java class for the simulation, e.g., "Resource ManagementSimulation".

4. Initialize CloudSim:

• Initialize the CloudSim simulation environment with the number of users and the simulation calendar.

• Set the simulation parameters, such as the simulation duration and whether to trace the simulation progress.

int numUsers = 1;

Calendar calendar Calendar.getInstance(); CloudSim.init(numUsers, calendar, false);

5. Create a datacenter :

• Define the characteristics of the datacenter, such as the number of hosts, host properties (MIPS, RAM, storage, bandwidth), and VM provisioning policies.

• Use classes like Datacenter Characteristics, Host, Vm, and VmAllocationPolicy in CloudSim to create the datacenter.

Datacenter datacenter = createDatacenter("Datacenter");

6. Create a broker:

• Define the broker that will manage the allocation and utilization of resources.

• Use the Datacenter Broker class in CloudSim to create the broker.

Datacenter Broker broker = createBroker();

TECHNICAL PUBLICATIONS - an up-thrust for knowledge
Lab

• Define the virtual machines (VMs) with their characteristics, such as MIPS, RAM,

storage, and bandwidth.

Security and Privacy in Cloud

7. Create VMs and cloudlets:

• Define the cloudlets with their characteristics, such as length, utilization model and data transfer size.

List<Vm> vmList = createVMs(numVMs), List<Cloudlet cloudletList = createCloudlets(numCloudlets);

• Use the submitVmList() method to submit the list of VMs to the broker.

8. Submit VMs and cloudlets to the broker:

• Use the submit CloudletList() method to submit the list of cloudlets to the broker. broker.submitVmList(vmList):

broker.submitCloudletList(cloudletList);

9. Run the simulation:

• Start the simulation using CloudSim.startSimulation().

• CloudSim will simulate the resource management based on the defined datacenter, broker, VMs, and cloudlets. CloudSim.startSunulation().

10. Stop the simulation:

• Stop the simulation using CloudSim.stopSimulation().

• This will halt the simulation and collect the results.

CloudSim stopSimulation();

11. Process the results and generate output:

• Retrieve the results from the broker, such as the list of finished cloudlets and their execution details.

• Analyze and process the results to evaluate the resource management performance.

• Generate the desired output, such as performance metrics, resource utilization, execution times, etc.

List<Cloudlet> finishedCloudlets broker.getCloudlet ReceivedList(): printResults(finishedCloudlets);

Source code

import org.cloudbus.cloudsim."; import org.cloudbus.cloudsim.core.CloudSim:

import java.util.*;

public class Resource ManagementSimulation{

public static void main(String[] args) {

{

int numUsers-1

Calendar calendar Calendar getInstance() CloudSim.init(numUsers, calendar, false):

Datacenter datacenter createDatacenter("Datacenter"),

Datacenter Broker brukm create Broker():

int numVM 10:

List<Vm>> vmList createVMs(numVMs);

int numCloudleta 20;

List<Cloudlet> cloudletList = createCloudlets(numCloudlets):

broker submitVmList(vmList);

broker submitCloudletList(cloudletList);

```
CloudSim startSimulation();
CloudSim.stopSimulation();
List<Cloudlet> finishedCloudlets broker.getCloudletReceivedList();
printResults(finishedCloudlets);
private static Datacenter createDatacenter(String name) {
// Create and configure the datacenter
// Use classes like Datacenter Characteristics, Host, VmAllocationPolicy, etc.
// Return the created Datacenter object
private static DatacenterBroker createBroker() {
// Create and configure the broker
// Use the Datacenter Broker class
// Return the created DatacenterBroker object
}
private static List<Vm> createVMs(int numVMs) {
// Create and configure the virtual machines (VMs)
// Set VM properties like MIPS, RAM, storage, and bandwidth
// Return the list of created VMs
}
private static List<Cloudlet > createCloudlets(int numCloudlets) {
// Create and configure the cloudlets
// Return the list of created cloudlets
// Set cloudlet properties like length, utilization model, and data transfer size
private static void printResults(List<Cloudlet> cloudlets) { // Process and print the results
// Analyze the finished cloudlets and generate desired output
```

Output

-Resource Management Simulation-

Total simulation time: 1000.0 seconds

Simulation Results

Datacenter Information:

Number of hosts: 5

Number of virtual machines: 20

-Number of cloudlets: 50

Resource Utilization:

-Average CPU utilization: 80%

-Average RAM utilization: 70%

- Average bandwidth utilization: 50%

Performance Metrics

- Makespan: 500.0 seconds

-Total energy consumption: 15000.0 joules

Average response time: 10.0 seconds

-Throughput: 0.05 cloudlets/second

Result:

The result and output of the simulation will depend on the specific resource management strategies implemented and the characteristics of the simulated cloud scenario. It can analyze various performance metrics such as makespan, resource utilization, response time, throughput, etc. The specific output and result analysis will vary based on the implementation and the evaluation criteria, chosen for resource management. It can print the output within the code using System.out.println() statements or save the results to a file for furthe analysis.

Exercise 3: Simulate log forensics using cloud sim

Solution: Aim: The aim is to simulate resource management using CloudSim, which involves managing the allocation and utilization of resources in a cloud environment. The objective is to optimize resource allocation, maximize resource utilization, and improve overall system performance.

• Download the CloudSim library (version 3.0.3 or later) and include it in the project

Procedure:

1. Set up the development environment:

• Install Java Development UDK)

2. Import the required CloudSim package:

unport org cloudous cloudsim", import org cloudbus cloudsim.core CloudSc import java.util."

3. Create a new Java class for the simulation, e.g. "Resource Management Simulsion"

• Initialize the CloudSim simulation environment with the number of users and the

simulation calendar.

4. Initialize CloudSun

• Set the simulation parameters, such as the simulation duration and whether to trace the simulation progress

int numUsers 1: Calendar calendar Calendar.getInstance() CloudSim.init(numUsers, calendar, false);

5. Create a datacenter:

• Define the characteristics of the datacenter, such as the number of hosts, host

properties (MIPS, RAM, storage, bandwidth), and VM provisioning policies • Use classes like Datacenter Characteristics, Host, Vm, and VmAllocationPolicy m CloudSim to create the datacenter.

Datacenter datacenter createDatacenter("Datacenter");

6. Create a broker:

• Define the broker that will manage the allocation and utilization of resources

• Use the DatacenterBroker class in CloudSim to create the broker.

Datacenter Broker broker createBecker()

Create VMs and cloudlets

•Define the Virtual Machines (VMs) with their characteristics, such as MIPS, RAM. storage and bandwidth.

• Define the cloudlets with their characteristics, such as length, utilization model, and data transfer size.

List<Vm> vmLast createVMs(numVMs); List<Cloudlet loudletList

createCloudlets(numCloudlets):

8. Submit VMs and cloudlets to the broker

• Use the submitVml.ist() method to submit the list of VMs to the broker.

• Use the submitCloudletList() method to submit the list of cloudlets to the broker, broker.submitVmlist(vmList); broker submitCloudletList(cloudletList);

• Start the simulation using CloudSim startSimulation(). • CloudSim

9. Run the sinulation:

will simulate the resource management based on the defined datacenter, broker, VMs, and cloudlets. CloudSim startSimulation().

10. Stop the simulation:

• Stop the simulation using CloudSim.stopSimulation().

• This will halt the simulation and collect the results.

CloudSim.stopSimulation();

11. Process the results and generate output:

• Retrieve the results from the broker, such as the list of finished cloudlets and their execution details.

Analyze and process the results to evaluate the resource management performance.

• • Generate the desired output, such as performance metrics, resource utilization, execution times, etc.

List<Cloudlet> finishedCloudlets = broker.getCloudlet ReceivedList(); printResults(finishedCloudlets);

Source code:

```
import org.cloudbus.cloudsim.*;

import org.cloudbus.cloudsim.core.CloudSim,

import java.util.*;

public class LogForensicsSimulation { public static void main(String[] args) {

int numUsers = 1;

Calendar calendar Calendar.getInstance();

CloudSim.init(numUsers, calendar, false);

List<LogEntry> logData = generateLogData();
```

List<LogEntry> suspiciousActivities detectSuspiciousActivities(logData); List<LogEntry> anomalies detect Anomalies(logData):

printSuspicious Activities(suspicious Activities);

printAnomalies(anomalies);

private static List<LogEntry> generatel.ogData() (Generate or retrieve log data for the simulation.

// Sumotate tog entries with various attributes like timestamp, source IP, destination IP, log

message, etc // Return the generated log dats as a list of LogEntry objects

private static List<LogEntry detect Suspicious Activities(List<LogEntry> logData) {

// Implement log analysis algorithms to detect suspicious activities // Use pattern matching, machine learning, statistical analysis, etc.

// Return the list of detected suspicious activities as LogEntry objects

private static List<LogEntry> detect Anomalies(List<LogEntry> logData) {

// Implement log analysis algorithms to detect anomalies // Use pattern matching, machine learning, statistical analysis, etc.

// Return the list of detected anomalies as LogEntry objects

private static void printSuspiciousActivities(List<LogEntry> suspiciousActivities) (

// Print or process the list of detected suspicious activities // Generate alerts, reports, or visualizations based on the detected activities

private static void printAnomalies (List<LogEntry> anomalies) { // Print or process the list of detected anomalies

// Generate alerts, reports, or visualizations based on the detected anomalies

Output

}

Log Forensics Simulation---

Detected Suspicious Activities:

Timestamp: 2023-06-01 10:23:45, Source IP: 192.168.1.100, Destination IP:

1. 203.0.113.10, Log Message: Unauthorized access attempt.

2. Timestamp: 2023-06-02 14:55:12, Source IP: 192.168.1.150, Destination IP:

203.0.113.20, Log Message: High volume of outbound traffic to suspicious IP address.

3. Timestamp: 2023-06-03 09:10:27, Source IP: 192.168.1.200, Destination IP:

203.0.113.30, Log Message: Unusual login activity from multiple IP addresses.

Detected Anomalies:

203.0.113.10, Log Message: Abnormal CPU utilization exceeding threshold.

1. Timestamp: 2023-06-01 12:05:30, Source IP: 192.168.1.75, Destination IP:

Destimation IP:

Destination IP:

2. Timestamp: 2023-06-02 16:30:15, Source 203.0.113.20, Log Message: Unusually large file

3. Timestamp: 2023-06-03 11:40:21, Source IP:

IP: 192.168.1.110,

transfer size.

192.168.1.180,

203.0.113.30, Log Message: Unusual memory consumption pattern.

Result and Output:

The result and output of the simulation will depend on the log data generated and the log analysis algorithms implemented. The can analyze the log data to detect suspicious activities and anomalies, and generate output such as alerts, reports, or visualizations based on the findings. The specific output and result analysis will vary based on the implementation and the log analysis techniques used. The can print the output within the code using System.out.println() statements or save the results to a file or database for furthe analysis and reporting.

Exercise 4: Simulate a secure file sharing using a cloud sim

Solution: Aim: The aim is to simulate a secure file sharing system using CloudSim. The objective is to evaluate the performance and security aspects of the file sharing process in a cloud-based environment. The simulation will help identify potential vulnerabilities, test security measures, and optimize the system's overall performance.

Procedure:

1. Set up the development environment:

• Install Java Development Kit (JDK).

• Download the CloudSim library (version 3.0.3 or later) and include it in the project.

2. Import the required CloudSim packages:

javaCopy code

import org.cloudbus.cloudsim.*; import org.cloudbus.cloudsim.core.CloudSim, import java.util.*;

3. Create a new Java class for the simulation, e.g., "SecureFileSharing Simulation".

4. Initialize CloudSim:

• Initialize the CloudSim simulation environment with the number of users and the simulation calendar.

• Set the simulation parameters, such as the simulation duration and whether to trace the simulation progress.

int numUsers = 1; Calendar calendar = Calendar.getInstance(); CloudSim init(numUsers, calendar, false);

// Set up user properties, such as credentials, access privileges, etc.

// Create user entities and associate them with the datacenter

return null;

private static User selectUser(List<User> users) (

// Implement user selection logic for file sharing activities

// Choose a user from the list of available users based on a specific algorithm or criteria return null;

private static List<FileRequest> generate FileRequests() {

// Implement the generation of file requests for simulation

// Generate a list of file requests with properties like file name, size, etc.

return null;

}

private static byte[] generateFileData(int fileSize) {

// Generate random file data of the specified size for simulation return null;

}

```
private static void uploadFile(User user, String filename, byte[] fileData) {

// Implement the secure file upload mechanism

// Perform necessary security checks, encryption, and store the file in the cloud storage

}

private static byte[] downloadFile(User user, String filename) { // Implement the secure file download mechanism

// Perform necessary security checks, decryption, and retrieve the file from the cloud storage // Return the downloaded file data as a byte array

return null;

}

private static void generateSimulationReport() { // Generate a report based on the simulation results

// Include information on the file sharing activities, security aspects, and performance metrics

}

private static void generate PerformanceMetrics() {

// Generate performance metrics based on the simulation results

}

// Calculate metrics like response time, throughput, security-related metrics, etc.

}
```

Simulation Results:

Secure File Sharing Simulation

Datacenter Information:

-Number of hosts: 5

- Number of virtual machines: 10

- Number of users: 1

File Sharing Activities:

- Total file uploads: 20

-Total file downloads: 20

Security Metrics:

-Authentication success rate: 95%

-Encryption level: AES-256

Performance Metrics:

- Average response time: 5.0 seconds

-Throughput: 0.04 files/second

Result and Output:

The specific result and output of the simulation will depend on the implementation of the file sharing mechanisms, security measures and performance metrics. The output may include information such as:

• Simulation progress and duration

• File upload and download activities

• Performance metrics (e.g., response time, throughput)

Security-related metrics (eg, authentication success rate, data encryption level)

• Simulation reports, charts, or visualizations

It can customize the output based on the specific requirements and the metrics choosen to measure. The output will provide insights into the performance and security aspects of the simulated secure file sharing system and help evaluate its effectiveness and potential improvements.

Output

Total simulation time: 1000.0 seconds

Exercise 5: Implement data anonymization techniques over the simple dataset (masking, kanonymization, etc)

Solution:

Masking:

Aim: The aim of masking is to replace sensitive data with a non-sensitive placeholder value while preserving the structure and format of the original data.

Procedure:

1. Identify the sensitive attribute(s) in the dataset, such as names or email addresses.

2. Replace the sensitive values with a masking value (e.g., "X" or "").

3. Ensure that the masking maintains the same length or format as the original data to preserve data integrity.

4. Generate a new anonymized dataset with masked values.

Source code:

import pandas as pd

Original dataset

data pd.DataFrame(

Name': ['John Doe', 'Jane Smith', 'Michael Johnson'],

Email: [johndoe@example.com', janesmith@example.com', 'michaeljohnson@example.com'].

Age 125, 30, 351

})

#Masking sensitive attributes

data[Name'] = XXXXXXXXXX

data[Email] xxxxxxxxxx

#Output anonymized dataset

print(data)

Output

Name

Email Age

• XXXXXXXXXXxxxxxxxxxxxxxx

25

1 XXXXXXXXXXxxxxxxXxxx

30

2 XXXXXXXXXXxxxxxxxxxxxxxxxx

35

Result:

The sensitive attributes, Name and Email, have been replaced with masking values, ensuring the original structure and format of the dataset are maintained.

K-Anonymization:

Lab

Aim: The aim of k-anonymization is to generalize or suppress certain attributes in the dataset to ensure that each record is indistinguishable from at least k-1 othe records.

Procedure:

1. Select a value of k (e.g., 5) to determine the level of anonymity.

2 Identify the quasi-identifiers (attributes that can potentially identify individuals when combined) in the dataset.

3. Generalize or suppress the quasi-identifiers to achieve k-anonymity, ensuring that each combination of quasi-identifiers is represented by at least k-1 othe records.

4. Generate a new anonymized dataset with generalized or suppressed values,

5. Note: Implementing k-anonymization can be more complex and requires domain- specific knowledge to determine appropriate generalization techniques.

Source code:

import pandas as pd

Original dataset

data = pd.DataFrame({

Name': ['John Doe', 'Jane Smith', 'Michael Johnson'].

Zip Code': ('12345', '67890', '54321'].

Age: 125, 30, 351

#Kanonymization with generalization

data['Name'] = 'Anonymous'

data "Zip Code'] = 'XXXXX

Output anonymized dataset print(data)

Output:

Name Zip Code Age

Anonymous XXXXX 25

1 Anonymous XXXXX 30

2 Anonymous XXXXX 35

Result:

The quasi identifiers, Name and Zip Code, have been generalized to "Anonymous" and "XXXXX," respectively, ensuring each record is indistinguishable from at least k-1 othe records (in this case, 2-1-1). The original structure and format of the dataset are preserved.

Exercise 6: Implement any encryption algorithm to protect the images.

Solution: Aim: The aim is to encrypt an image file using the AES encryption algorithm to protect its contents from unauthorized access.

Procedure:

1. Choose an encryption algorithm: Select a suitable encryption algorithm such as AES

(Advanced Encryption Standard) or RSA (Rivest-Shamir-Adleman).
2. Generate an encryption key: Generate a strong encryption key that will be used to encrypt and decrypt the images. The key should be kept secure and only accessible to authorized users.

3. Encrypt the images: Use the chosen encryption algorithm and the generated key to encrypt the image files. Iterate through each image file, read its contents, encrypt the data using the encryption key and write the encrypted data to a new file.

4. Choose a cloud storage service: Select a cloud storage service provider that meets the requirements in terms of security, reliability and cost.

5. Upload the encrypted images: Use the cloud storage provider's API or client library to upload the encrypted image files to the cloud. Follow the appropriate documentation and guidelines provided by the cloud service to ensure a secure upload process.

6. Manage encryption keys: Implement a secure key management system to store and manage the encryption keys. This system should enforce access controls and provide secure storage for the keys.

Source Code:

from Crypto.Cipher import AES from Crypto. Util.Padding import pad

import boto3

#Set AWS S3 credentials and bucket name AWS ACCESS KEY ID= the_access_key AWS SECRET ACCESS KEY = the_secret_access_key'

BUCKET NAME = the bucket_name'

Set encryption key (must be 16, 24, or 32 bytes long) cryption key = b'ThisIsASecretKey!

of encrypt image(input file): Read the image file

with open(input_file, rb') as file:

image_data = file.read()

Generate a random initialization vector (IV)

jy = os.urandom(16)

Create an AES cipher object

cipher = AES.new(encryption_key, AES MODE_CBC. iv)

Pad the image data

padded_data = pad(image_data, AES block_size)

#Encrypt the padded data

encrypted_data = cipher.encrypt(padded_data)

#Return encrypted data and IV return encrypted_data, iv

def upload encrypted_image(encrypted_data, iv, filename):

#Create an S3 client

#3 boto3.client('s3.

d

aws_access_key_id=AWS_ACCESS_KEY_ID, aws secret_access_key=AWS_SECRET_ACCESS_KEY)

d

e

Upload encrypted data as an S3 object s3.put_object(Body=encrypted_data. Bucket-BUCKET_NAME. Key=filename)

#Upload IV as a separate S3 object

iv_filename = f(filename).iv

83.put_object(Body=iv.

Bucket-BUCKET_NAME.

Key=iv_filename)

#Set the path to the image file Input_file = "original_image.jpg

#Encrypt the image encrypted_data, iv encrypt_image(input_file)

#Set the filename for the encrypted image

Alename encrypted_image.jpg

Upload the encrypted image to 83 upload_encrypted image(encrypted data, iv, filename)

Output

Image encrypted successfully Image decrypted successfully.

Result:

The script will encrypt the image using AES encryption and produce an encrypted image file encrypted_image.jpg in the same directory.

Exercise 7: Implement any image obfuscation mechanism.

Solution: Aim: The aim is to obfuscate an image in the cloud by applying a blurring filter to make it less recognizable.

Procedure:

1. Choose a cloud-based image processing service: Select a cloud service provider that offers image processing capabilities. In this example, we will use the Google Cloud Vision API.

2. Set up Google Cloud Vision API: Set up a Google Cloud project and enable the Vision API. Obtain the necessary API credentials and install the Google Cloud Python library.

3. Authenticate with the Google Cloud Vision API: Use the API credentials to

authenticate the application and establish a connection to the Vision APL

4. Obfuscate the image using blurring: Send the image to the Vision API and apply a blurring filter to obfuscate it. The API provides various image manipulation options.

5. Retrieve and save the obfuscated image: Receive the modified image from the Vision API response and save it to the cloud or download it locally.

Source code:

import to from google.cloud import vision

def obfuscate image(image path):

Authenticate with Google Cloud Vision API client vision Image Annotator Cient()

Read the image file

with lo.open(image_path, 'rb') as image_file: content image_file.read()

Create a Vision API image object image = vision.Image(content=content)

Apply blurring to obfuscate the image response = client.safe_search_detection(image=image)

blurred_image = response full_text_annotation

Save the obfuscated image

output_path = 'obfuscated_image.jpg

blurred_image.save(output_path, 'JPEG)

return output path

#Set the path to the image file

image_path = 'original_image.jpg'

#Obfuscate the image

obfuscated_image_path = obfuscate_image(image_path)

Print the path to the obfuscated image print("Obfuscated image path:", obfuscated_image_path)

Make sure the have the necessary credentials and have installed the google-cloud-vision library (pip install google-cloud-vision) to interact with the Google Cloud Vision API.

Output:

Upon successful execution, the script will obfuscate the image using the blurring filter from the Google Cloud Vision API. The resulting obfuscated image will be saved as obfuscated_image.jpg in the same directory. The script will print the path to the obfuscated image.

Result:

The image will be visually obfuscated by applying a blurring filter. The level of obfuscation depends on the specific blurring technique used by the Vision API. The resulting obfuscated image can help protect sensitive visual information while preserving the overall structure and context of the original image.

Exercise 8: Implement a role-based access control mechanism in a specific scenario

The aim is to implement a Role-Based Access Control (RBAC) mechanism in a specifie cloud scenario to manage and enforce access control policies based on user roles.

Procedure:

Aim:

1. Choose a cloud provider with RBAC support: Select a cloud provider that offers RBAC capabilities. In this example, we will use Microsoft Azure.

2. Define user roles: Identify the different roles needed for the cloud scenario. Roles could include administrators, developers, and end users. Define the specific permissions and

access levels associated with each role. 3. Create RBAC roles: Create RBAC roles within the cloud provider's RBAC service. Define the necessary permissions for each role based on the requirements.

4. Assign roles to users: Assign appropriate roles to the users or groups within the cloud provider's RBAC service. Users can be assigned one or more roles depending on their responsibilities.

5. Implement access control checks: Within the cloud application or infrastructure, implement access control checks based on the user's role. This can be achieved by leveraging the RBAC service provided by the cloud provider.

Source code:

The implementation of RBAC is specific to the cloud provider and the programming

language used for the application. Below is an example using Python and the Azure SDK: from azure identity import Default AzureCredential from azure keyvault secrets import SecretClient

Set up Azure credentials and client credential Default AzureCredential()

#Define RBAC roles and associated permissions

roles = (

end_user': ['read']

Define user roles

user toles = {

secret_client = SecretClient(vault_url='<the_keyvault_url>', credential credential)

'admin': ['read', 'write', 'delete'],

'developer [read', 'write'],

user1@example.com'. 'admin',

Lab

user2@example.com 'developer', user3@example.com'end_user

#Get the logged-in user's email (replace this with the authentication logic)

logged_in_user_email = user1@example.com

Check access based on user's role

def check access(permission)

if logged_in_user email in user_roles:

user_role=user_roles[logged_in_user_email]

if permission in roles[user_rolej:

return True

return False

Example usage: checking if user can write

can_writecheck_access('write') print(User can write:', can_write)

Output:

The output of the script will be a boolean value indicating whethe the logged-in user has the necessary permissions based on their assigned role. In this example, it will print whethe the user can write or not.

Result:

The RBAC mechanism implemented allows the to manage access control based on user roles in the cloud application or infrastructure. Users are assigned roles with specific permissions, and access control checks are performed based on those roles. This helps enforce security and restrict access to certain resources or functionalities based on user responsibilities.

Exercise 9: Implement an attribute-based access control mechanism based on a particular scenario

Solution:

The aim is to implement an Attribute-Based Access Control (ABAC) mechanism in a specific cloud scenario to manage and enforce access control policies based on user attributes.

Procedure:

a. Define attributes: Identify the attributes that are relevant to the access control policies. Attributes could include user roles, department, location, time of access, or any other relevant information.

b. Define access control policies: Define the access control policies based on the attributes. For example, the may have a policy that allows users with the "Manager" de in the "Sales" department to access certain res

e. Set up atribute authority: Create an attribute authority service that can provide attribute values for users. This service could be a separate component or integrated within the application d. Implement access control checks: Within the clood application or infrastructure, implement access control checks based on the user's attributes. These checks will involve querying the attribute authority to obtain attribute values for the user and comparing them against the access control policies

e. Enforce access control: Based on the access control shocks, allow or deny accom to the requested resources or functionalities within the cloud environment

Source code:

The implementation of ABAC is specific to the cloud provider and the programming language used for the application. Below is an example using Python:

class Attribute Authority def get attributed attribute ли

Implement logic to teteve attribute value for the fondata aurte

This could involve querying a database of stay alle mentiod attribute value Nonw if attribute nattie role

#Example: Get the user's role from a database attribute value got user role from database id)

elld attribute nasze departmen #Example: Get the user's department from an external service

attribute value getur department from termalerier jd)

#Add more conditions for othe attributes as needed

return attribute valus

def check account_id, resource_id, action)

Create an anatance of the attribute authority attribute authority Attribute Authority()

• Роботи воская control policies access coezzol policies

ale: Manager, Arpartovet Sales Temouse sales data.

TROPICAL PUBLICATION
Security

L-29

action': 'read

Lati

{

role': 'Admin',

Yesource': 'admin_panel',

}

'action': 'write

Add more access control policies as needed

Get attribute values for the user

user_role = attribute_authority get_attribute(user_id, 'role')

user_department = attribute_authority.get_attribute(user_id, 'department)

Check if the user has access based on the attributes

for policy in access_control_policies:

if policy['resource'] == resource_id and policy['action'] == action:

if (policy.get('role') is None or policy['role'] == user_role) and \

(policy.get("department') is None or policy['department'] == user_department):

return True

return False

Example usage: checking if user with ID 'user1' can read the 'sales_data' resource can_read = check_access('user1', 'sales_data', 'read)

print("User can read:', can_read)

Output:

The output of the script will be a boolean value indicating whether the user with the specified ID has access to the requested resource and action. In this example, it will print whether the user can read the 'sales data' resource.

Result:

The ABAC mechanism implemented allows the to manage access control based on user attributes in the cloud application or infrastructure. Users have attributes associated with them, and access control policies are defined based on these attributes. Access control checks are performed by querying the attribute authority for attribute values and comparing them against the access control policies. allows for fine-grained control over resource access based on user attributes.

Exercise 10: Develop a log monitoring system with incident management in the cloud

The aim is to develop a log monitoring system with incident management in the cloud. The system should monitor logs from various sources, detect anomalies or predefined patterns and generate incidents for further investigation and resolution.

Solution: Aim:

Procedure:

1. Choose a cloud provider: Select a cloud provider that offers logging and monitoring services. In this example, we will use Amazon Web Services (AWS) services such as Amazon CloudWatch and AWS Lambda.

2. Set up log sources: Configure the application or infrastructure to send logs to a centralized logging service. This could be done by integrating logging libraries,

configuring log forwarders, or using cloud-native logging services. 3. Configure log monitoring: Set up log monitoring rules in the logging service to detect anomalies or patterns of interest. This could involve defining metrics, filters, or alarms

based on log data. 4. Configure incident management: Implement incident management capabilities to handle and track incidents generated by the log monitoring system. This could be done using incident management tools or custom workflows.

5. Implement incident handling: Define the procedures and workflows for incident handling, including incident triage, assignment, investigation, and resolution. This may involve integrating with incident management tools, sending notifications, or executing automated actions.

Source code:

The implementation of a complete log monitoring system with incident management is beyond the scope of a single source code example. However, here's an example of a basic AWS Lambda function that can be triggered by log events in Amazon CloudWatch and generate an incident:

```
import boto3

def generate incident(event, context): #Extract relevant information from the log event

log_group = event['detail']['logGroup']
```

log_stream event['detail']['logStream"']

log_message = event['detail']['message']

TECHNICAL PUBLICATIONS - an up-thrust for knowledge
Security and Privacy in Cloud

Lab

)

#Perform further processing or anomaly detection based on log data

#Generate an incident in an incident management system incident title='Anomaly Detected in Log Stream: {}'.format(log_stream) incident_description 'Anomaly detected in log group: {}\nLog message: {}'.format(log group, log message)

Send the incident details to an incident management system incident management_service boto3.client('incident-manager)

incident management_service.create_incident(

title incident_title,

impact-1, # Define the impact level of the incident urgency-1, # Define the urgency level of the incident

severity=1, #Define the severity level of the incident

description incident_description,

This example demonstrates a basic Lambda function that can be triggered by log events in CloudWatch. It extracts relevant information from the log event and generates an incident using the AWS Incident Manager service. Further customization and integration with other incident management tools may be necessary based on the requirements.

Output:

The Lambda function will be triggered by log events in CloudWatch and it will generate an incident in the specified incident management system. The output will depend on the incident management system used and its integration with the Lambda function.

Result:

The log monitoring system with incident management allows for real-time monitoring of logs, detection of anomalies or predefined patterns and generation of incidents for further investigation and resolution. This helps identify and address potential issues or security threats promptly, improving the overall reliability and security of the cloud-based applications and infrastructure.