Implement any encryption algorithm to protect the images

package cryptog;

import java.io.\*;

import java.security.\*;

import javax.crypto.\*;

import javax.crypto.spec.SecretKeySpec;

public class cryptog {

public static void main(String[] args) {

String inputFile = "C:\\spc programs\\image.png";

String encryptedFile = "C:\\spc programs\\encrypted\_image.jpg";

String decryptedFile = "C:\\spc programs\\decrypted\_image.jpg";

// Generate AES key

byte[] keyBytes = new byte[16];

SecureRandom secureRandom = new SecureRandom();

secureRandom.nextBytes(keyBytes);

SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");

try {

// Encrypt image

encrypt(inputFile, encryptedFile, key);

// Decrypt image

decrypt(encryptedFile, decryptedFile, key);

System.out.println("message encrypted and decrypted successfully");

} catch (Exception e) {

e.printStackTrace();

}

}

public static void encrypt(String inputFile, String outputFile, SecretKey key) throws Exception {

Cipher cipher = Cipher.getInstance("AES");

cipher.init(Cipher.ENCRYPT\_MODE, key);

try (FileInputStream fis = new FileInputStream(inputFile);

FileOutputStream fos = new FileOutputStream(outputFile);

CipherOutputStream cos = new CipherOutputStream(fos, cipher)) {

byte[] buffer = new byte[4096];

int bytesRead;

while ((bytesRead = fis.read(buffer)) != -1) {

cos.write(buffer, 0, bytesRead);

}

}

}

public static void decrypt(String inputFile, String outputFile, SecretKey key) throws Exception {

Cipher cipher = Cipher.getInstance("AES");

cipher.init(Cipher.DECRYPT\_MODE, key);

try (FileInputStream fis = new FileInputStream(inputFile);

FileOutputStream fos = new FileOutputStream(outputFile);

CipherInputStream cis = new CipherInputStream(fis, cipher)) {

byte[] buffer = new byte[4096];

int bytesRead;

while ((bytesRead = cis.read(buffer)) != -1) {

fos.write(buffer, 0, bytesRead);

}

}

}

}

Implement any image obfuscation mechanism.

package cryptog;

import javax.imageio.ImageIO;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

public class obfus {

public static void main(String[] args) {

String inputImagePath = "C:\\spc programs\\image1.jpg";

String outputImagePath = "C:\\spc programs\\obfuscated\_image.jpg";

try {

BufferedImage inputImage = ImageIO.read(new File(inputImagePath));

BufferedImage obfuscatedImage = obfuscateImage(inputImage);

File outputFile = new File(outputImagePath);

ImageIO.write(obfuscatedImage, "jpg", outputFile);

System.out.println("Image obfuscated successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

public static BufferedImage obfuscateImage(BufferedImage image) {

int width = image.getWidth();

int height = image.getHeight();

// Create a new BufferedImage with the same dimensions and type as the original image

BufferedImage obfuscatedImage = new BufferedImage(width, height, image.getType());

// Iterate over each pixel in the image

for (int y = 0; y < height; y++) {

for (int x = 0; x < width; x++) {

// Get the RGB values of the pixel

int rgb = image.getRGB(x, y);

// Apply a simple transformation to obfuscate the pixel

int red = (rgb >> 16) & 0xFF;

int green = (rgb >> 8) & 0xFF;

int blue = rgb & 0xFF;

// Invert the colors

red = 255 - red;

green = 255 - green;

blue = 255 - blue;

// Combine the modified RGB values to form the new pixel

int obfuscatedRGB = (red << 16) | (green << 8) | blue;

// Set the pixel in the obfuscated image

obfuscatedImage.setRGB(x, y, obfuscatedRGB);

}

}

return obfuscatedImage;

}

}

Implement a role-based access control mechanism in a specific scenario.

**package** cryptog;

**import** java.util.\*;

//Role class represents a role in the system

**class** Role {

**private** String roleName;

**private** Set<String> permissions;

**public** Role(String roleName) {

**this**.roleName = roleName;

**this**.permissions = **new** HashSet<>();

}

**public** String getRoleName() {

**return** roleName;

}

**public** **void** addPermission(String permission) {

permissions.add(permission);

}

**public** **boolean** hasPermission(String permission) {

**return** permissions.contains(permission);

}

}

//User class represents a user in the system

**class** User {

**private** String username;

**private** Set<Role> roles;

**public** User(String username) {

**this**.username = username;

**this**.roles = **new** HashSet<>();

}

**public** String getUsername() {

**return** username;

}

**public** **void** addRole(Role role) {

roles.add(role);

}

**public** **boolean** hasPermission(String permission) {

**for** (Role role : roles) {

**if** (role.hasPermission(permission)) {

**return** **true**;

}

}

**return** **false**;

}

}

**public** **class** RBACExample {

**public** **static** **void** main(String[] args) {

// Create roles

Role adminRole = **new** Role("Admin");

adminRole.addPermission("READ");

adminRole.addPermission("WRITE");

Role userRole = **new** Role("User");

userRole.addPermission("READ");

// Create users

User adminUser = **new** User("admin");

adminUser.addRole(adminRole);

User normalUser = **new** User("user");

normalUser.addRole(userRole);

// Test permissions

*testPermissions*(adminUser);

*testPermissions*(normalUser);

}

**private** **static** **void** testPermissions(User user) {

System.***out***.println("User: " + user.getUsername());

System.***out***.println("Has READ permission: " + user.hasPermission("READ"));

System.***out***.println("Has WRITE permission: " + user.hasPermission("WRITE"));

System.***out***.println();

}

}

Implement an attribute-based access control mechanism based on a particular scenario.

package cryptog;

import java.util.HashMap;

import java.util.Map;

// User class represents a user with attributes

class User1 {

private String username;

private Map<String, String> attributes;

public User1(String username) {

this.username = username;

this.attributes = new HashMap<>();

}

public String getUsername() {

return username;

}

public void addAttribute(String name, String value) {

attributes.put(name, value);

}

public String getAttribute(String name) {

return attributes.get(name);

}

}

// Resource class represents a resource with attributes

class Resource {

private String name;

private Map<String, String> attributes;

public Resource(String name) {

this.name = name;

this.attributes = new HashMap<>();

}

public String getName() {

return name;

}

public void addAttribute(String name, String value) {

attributes.put(name, value);

}

public String getAttribute(String name) {

return attributes.get(name);

}

}

// ABACPolicy class represents an ABAC policy

class ABACPolicy {

private String subjectAttribute;

private String resourceAttribute;

private String actionAttribute;

private String effect;

public ABACPolicy(String subjectAttribute, String resourceAttribute, String actionAttribute, String effect) {

this.subjectAttribute = subjectAttribute;

this.resourceAttribute = resourceAttribute;

this.actionAttribute = actionAttribute;

this.effect = effect;

}

public String getSubjectAttribute() {

return subjectAttribute;

}

public String getResourceAttribute() {

return resourceAttribute;

}

public String getActionAttribute() {

return actionAttribute;

}

public String getEffect() {

return effect;

}

public boolean evaluate(User1 user, Resource resource, String action) {

String subjectValue = user.getAttribute(subjectAttribute);

String resourceValue = resource.getAttribute(resourceAttribute);

String actionValue = action;

// Implement policy evaluation logic based on attribute values

// Here, we assume that if any attribute value is null or does not match, deny access

if (subjectValue == null || resourceValue == null || actionValue == null) {

return false;

}

// For simplicity, we only consider the effect "allow"

return subjectValue.equals(resourceValue) || actionValue.equals(actionAttribute) && effect.equals("allow");

}

}

public class ABACExample {

public static void main(String[] args) {

// Create users with attributes

User1 alice = new User1("alice");

alice.addAttribute("department", "engineering");

alice.addAttribute("role", "engineer");

User1 bob = new User1("bob");

bob.addAttribute("department", "sales");

bob.addAttribute("role", "salesperson");

// Create resources with attributes

Resource documentA = new Resource("documentA");

documentA.addAttribute("department", "engineering");

Resource documentB = new Resource("documentB");

documentB.addAttribute("department", "sales");

// Define ABAC policies

ABACPolicy policy1 = new ABACPolicy("role", "department", "read", "allow");

ABACPolicy policy2 = new ABACPolicy("role", "department", "write", "denied");

// Test access

testAccess(alice, documentA, "read", policy1);

testAccess(alice, documentA, "write", policy2);

testAccess(bob, documentA, "read", policy1);

testAccess(bob, documentA, "write", policy2);

testAccess(alice, documentB, "read", policy1);

testAccess(alice, documentB, "write", policy2);

testAccess(bob, documentB, "read", policy1);

testAccess(bob, documentB, "write", policy2);

}

private static void testAccess(User1 user, Resource resource, String action, ABACPolicy policy) {

boolean access = policy.evaluate(user, resource, action);

System.out.println("User " + user.getUsername() + " attempting to " + action + " " + resource.getName() + ": " + (access ? "Allowed" : "Denied"));

}

}

Develop a log monitoring system with incident management in the cloud.

package cryptog;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

public class logmonitor {

// Define thresholds for incident detection

private static final int ERROR\_THRESHOLD = 5;

private static final int WARNING\_THRESHOLD = 3;

// Map to store log counts

private static Map<String, Integer> logCounts = new HashMap<>();

public static void main(String[] args) {

// Replace this path with the actual path to your log file

String logFilePath = "/path/to/logfile.log";

// Continuously monitor the log file

while (true) {

try {

// Read the log file

readLogFile(logFilePath);

// Check for incidents

checkIncidents();

// Sleep for 1 minute (adjust as needed)

Thread.sleep(60000);

} catch (InterruptedException | IOException e) {

e.printStackTrace();

}

}

}

// Method to read the log file

private static void readLogFile(String filePath) throws IOException {

File file = new File(filePath);

BufferedReader reader = new BufferedReader(new FileReader(file));

String line;

while ((line = reader.readLine()) != null) {

// Process each log line

processLogLine(line);

}

reader.close();

}

// Method to process each log line

private static void processLogLine(String line) {

// Assuming each log line contains a log level (e.g., ERROR, WARNING)

String logLevel = line.split(" ")[0];

// Increment the count for the log level

logCounts.put(logLevel, logCounts.getOrDefault(logLevel, 0) + 1);

}

// Method to check for incidents

private static void checkIncidents() {

int errorCount = logCounts.getOrDefault("ERROR", 0);

int warningCount = logCounts.getOrDefault("WARNING", 0);

// Check if error count exceeds threshold

if (errorCount >= ERROR\_THRESHOLD) {

// Trigger incident management for errors

handleIncident("ERROR", errorCount);

}

// Check if warning count exceeds threshold

if (warningCount >= WARNING\_THRESHOLD) {

// Trigger incident management for warnings

handleIncident("WARNING", warningCount);

}

}

// Method to handle incidents

private static void handleIncident(String level, int count) {

// Replace this with your incident management logic (e.g., send notification)

System.out.println("Incident detected: " + level + " count exceeds threshold (" + count + ")");

}

}