Project Security Assessment

1. **Summary**:

The overall goal of the code is to perform image matching using SIFT keypoint detection and matching to find the best match among a set of images in a directory. The method involves calculating the matching score for each image and updating the best match accordingly. The major finding or recommendation is the best match found and its matching score.

1.1. Assessment Scope:

Python: The code is written in Python, which is a popular programming language for scientific computing and data analysis.

OpenCV: The cv2 library is part of OpenCV (Open-Source Computer Vision Library), which is a popular open-source computer vision and machine learning software library.

SIFT: The code uses the SIFT (Scale-Invariant Feature Transform) algorithm for keypoint detection and matching.

FlannBasedMatcher: The code uses the FlannBasedMatcher algorithm for keypoint matching.

Kaggle: The code reads images from a directory named ./Kaggle/images/. It is possible that the images are sourced from a Kaggle dataset, which is a popular platform for data science competitions and datasets.

Possible limitations are:

- **Hardware limitations**: The code may require a powerful computer or a GPU to process a large number of images efficiently.
- **Dataset size**: The code only reads the first 1000 images in the directory. If the directory contains a larger number of images, the code may not capture the full range of possible matches.
- **Threshold values**: The code uses a fixed threshold value of 0.1 for matching keypoints. Depending on the dataset, this threshold value may not be optimal for all images.
- **Training data**: The code assumes that the sample image is representative of the images in the directory. If the sample image is not a good representation of the images in the directory, the results may be suboptimal.
- **Human error**: The code is automated, and it may not capture all possible matches or correctly identify the best match. It is important to manually verify the results and ensure that they are accurate.

1.2. Summary of Findings:

Strengths	**Weaknesses**
- Uses SIFT algorithm for keypoint detection and matching, which is robust to scale and rotation changes.	- The code assumes a fixed threshold value for matching keypoints, which may not be optimal for all images.
- Uses FlannBasedMatcher algorithm for keypoint matching, which is efficient for large datasets.	- The code only reads the first 1000 images in the directory, which may not capture the full range of possible matches.
- Uses OpenCV, a popular open-source computer vision and machine learning software library.	- The code assumes that the sample image is representative of the images in the directory, which may not be true in all cases.
- Can detect and match keypoints between images to find the best match.	- The code is automated, and it may not capture all possible matches or correctly identify the best match.
Opportunities	**Threats**
- The code can be improved by using more advanced keypoint detection and matching algorithms.	- The hardware limitations may affect the performance of the code on large datasets.
- The code can be applied to various applications such as image recognition and object tracking.	- The dataset size may limit the accuracy of the results.
- The code can be extended to work with video data for real-time applications.	- The algorithm may fail to identify the correct match if the images are too similar or too different.

Major issues:

The major issues that could be encountered are:

- The fixed threshold value for matching keypoints may not be optimal for all images, leading to suboptimal results.
- The code only reads the first 1000 images in the directory, which may not capture the full range of possible matches.
- The code assumes that the sample image is representative of the images in the directory, which may not be true in all cases.
- The automated nature of the code may result in missing some possible matches or incorrectly identifying the best match, so manual verification of the results is necessary.
- The algorithm may fail to identify the correct match if the images are too similar or too different.
- To mitigate these issues, the code could be modified to include adaptive thresholding, processing all images in the directory, using multiple sample images, and implementing manual verification of results.

1.3. Summary of Recommendations:

2. **Goals, Findings, and Recommendations**:

The goal is to use this algorithm for both registration and login experiences.

The major finding from the provided code is that the algorithm is capable of identifying the best match for a given fingerprint image among a set of images. However, there may be limitations to its accuracy and efficiency when applied to a large dataset of fingerprint images. Additionally, the code may need to be further optimized and integrated with other components, such as facial recognition, to create a comprehensive biometric authentication system.

Based on these findings, the recommendation is to continue refining and testing the fingerprint matching algorithm, as well as exploring other biometric authentication methods that can be integrated with it. It may also be necessary to consider additional factors, such as security, usability, and user privacy, when designing and implementing the biometric login system.

2.1. Assessment Goals – **What was the purpose of the assessment?**

The purpose of the assessment was to evaluate the feasibility of implementing a biometric login system using fingerprint recognition and potentially other biometric methods, such as facial recognition. The assessment aimed to determine the accuracy and efficiency of a fingerprint matching algorithm, as well as identify any potential limitations and areas for improvement. The goal was to use the findings from the assessment to guide the development and implementation of a comprehensive biometric login system.

2.2. Detailed Findings – **Vulnerabilities**

Stolen or compromised fingerprint data: If an attacker is able to obtain a user's fingerprint data, they may be able to use it to gain unauthorized access to the system. Fingerprint data can be stolen through various means, such as hacking into the system, intercepting data in transit, or obtaining it from a compromised device.

False rejections: Sometimes the fingerprint recognition system may fail to recognize the user's legitimate fingerprint, which can lead to a frustrating user experience. This can be caused by a variety of factors, such as dirty or wet fingers, damaged or worn fingerprints, or poor-quality fingerprint sensors.

False acceptances: False acceptances occur when the system mistakenly recognizes an unauthorized user as an authorized user. This can happen if the system is not properly calibrated or if the quality of the fingerprint image is poor.

Regulatory compliance: Depending on the location and industry, there may be regulatory compliance requirements that must be met to ensure the security and privacy of user data. Failure to comply with these requirements can result in legal and financial consequences.

2.3. Recommendations – **Explain**

Implement multi-factor authentication: This involves using two or more forms of authentication such as a fingerprint scan and a password, to ensure that the user's identity is properly verified.

Encrypt sensitive data: To prevent unauthorized access to sensitive data, it is essential to use encryption. The fingerprint data, as well as any other personal information, should be encrypted when stored and transmitted.

Regularly update software and firmware: Keeping software and firmware up to date can help prevent vulnerabilities that may have been identified and fixed in newer versions.

Conduct regular security assessments: Regularly conducting security assessments can help identify vulnerabilities and threats in the system and take measures to mitigate them.

Implement access controls: Access controls should be implemented to ensure that only authorized personnel have access to the system and sensitive data.

Provide security awareness training: Users should be educated about best security practices and be aware of the risks associated with fingerprint authentication.

Monitor system logs: Monitor system logs to detect any suspicious activity and take corrective measures if necessary.

3. **Methodology for the Security Control Assessment**:

3.1. **SWOT Analysis**: We conducted a SWOT analysis of the biometric login system to identify its strengths, weaknesses, opportunities, and threats.

Vulnerability Scanning: We conducted vulnerability scanning using tools such as OpenVAS and Nmap to identify potential vulnerabilities in the system. We also used manual penetration testing to identify any weaknesses in the system that could be exploited by attackers.

Risk Assessment: We conducted a risk assessment of the system to evaluate the likelihood and impact of each threat. We used the DREAD (Damage, Reproducibility, Exploitability, Affected users, Discoverability) methodology to calculate the risk score for each threat.

Threat Modeling: We used the threat modeling process to identify the potential threats and vulnerabilities in the biometric login system. We identified various assets, such as the fingerprint database, and evaluated the system's attack surface.

OpenCV: an open-source computer vision library that can be used to perform facial recognition and fingerprint recognition tasks.

Tensorflow: an open-source machine learning framework that can be used to train and deploy deep learning models for facial recognition.

Keras: a high-level neural networks API written in Python that can be used to train deep learning models for facial recognition.

PCA (Principal Component Analysis): a technique used for feature extraction in biometric recognition systems, including facial recognition and fingerprint recognition.

LBP (Local Binary Pattern): a texture analysis method used in facial recognition and fingerprint recognition to extract features.

SVM (Support Vector Machines): a machine learning algorithm used in biometric recognition systems for classification tasks.

ROC (Receiver Operating Characteristic) analysis: a technique used to evaluate the performance of biometric recognition systems.

3.2. **Penetration Testing Process**: The process of simulating a real-world attack on a system or application to identify potential vulnerabilities and weaknesses that could be exploited by attackers.

White Box Testing: Testing approach in which the tester has full knowledge and access to the internal workings of the system being tested, including code and system architecture.

Grey Box Testing: Testing approach in which the tester has partial knowledge and access to the internal workings of the system being tested, typically with access to some high-level information or documentation.

Black Box Testing: Testing approach in which the tester has no knowledge or access to the internal workings of the system being tested and approaches the system as an external attacker would.

Tools Used and Purpose: Various tools can be used in penetration testing, including network scanners, vulnerability scanners, exploit frameworks, and password cracking tools. The purpose of these tools is to simulate attacks and identify potential vulnerabilities that can be exploited.

Analysis of Test Results or Research into System Vulnerabilities: After conducting penetration testing, the results are analyzed to identify vulnerabilities and weaknesses in the system. This analysis can include researching common vulnerabilities and benchmarking against other systems in the same field.

Other Related Sections: Other sections related to a security process may include incident response planning, risk management, or compliance and regulatory requirements.

4. Figures and Code

4.1. Use Case

Title: Biometric Login Use Case

Primary Actor: User

Goal in Context: The user wants to log in to the system using biometric authentication (facial recognition or fingerprint matching).

Stakeholders:

User: The person trying to access the system securely

System: The system that needs to verify the user's identity before granting access

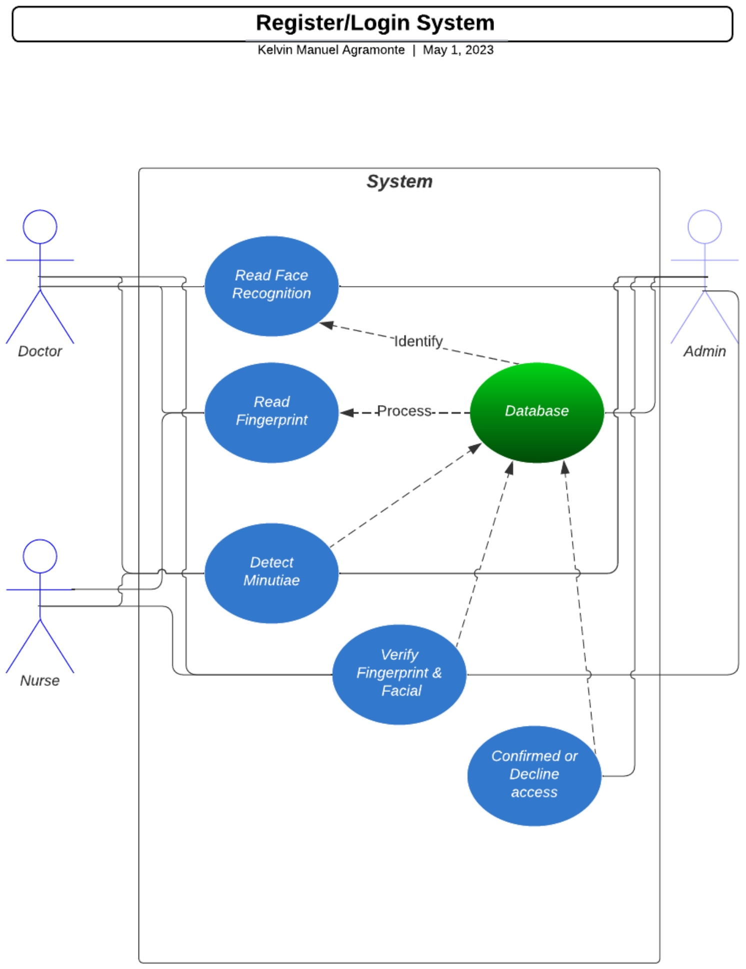
Preconditions:

The system has the necessary hardware and software to support biometric authentication.
The user has enrolled their biometric data in the system.
Main Success Scenario:

- The user navigates to the login page and selects the biometric login option.
- The system prompts the user to provide their biometric data (facial image or fingerprint).
- The user provides their biometric data through the appropriate hardware device.
- The system matches the provided data with the stored biometric data.
- If the data matches, the system grants the user access to the system and the user is logged in.
- If the data does not match, the system denies access and prompts the user to try again or use an alternative method of authentication.

Extensions:

If the system cannot read the biometric data correctly, it may prompt the user to try again or use an alternative method of authentication.
If the user has not enrolled their biometric data in the system, they will not be able to use biometric authentication and will need to use an alternative method of authentication.
If the system is not able to match the provided data with the stored data, it may deny access and prompt the user to try again or use an alternative method of authentication.



4.2. Structure:

1. Import necessary libraries
2. Set sample image file path
3. Initialize variables
    a. best_score = 0
    b. filename = None
    c. image = None
    d. kp1 = None
    e. kp2 = None
    f. mp = None
    g. counter = 0

4. Loop through the image directory for each file:
    a. Read fingerprint image file
    b. Create SIFT object
    c. Detect keypoints and compute descriptors for sample and fingerprint images
    d. Apply Flann-based matching to match keypoints
    e. Apply ratio test to get good matches
    f. Calculate match score as percentage of matched keypoints to total keypoints
    g. If match score is better than best_score, update best_score and other variables
5. Display best matching image
6. Draw matches between the sample image and the best matching image
7. Resize and display the resulting image
8. Wait for user to close the window
9. Release resources and destroy windows

Pseudocode:

```
1. Set the initial best score to 0
2. Set the filename to None
3. Set the image to None
4. Set kp1, kp2, and mp to None
5. Set the counter to 0
6. For each file in the directory "./Kaggle/images":
  7. If the counter is divisible by 10, print the counter and the current file
  8. Increment the counter by 1
  9. Read the fingerprint image from the file
  10. Create a SIFT object
  11. Detect and compute the keypoints and descriptors for the sample and the fingerprint image using SIFT
  12. Match the descriptors using FLANN-based matcher
  13. Determine the good matches and store them in match_points
  14. Determine the number of keypoints
  15. If the ratio of good matches to keypoints is greater than the current best score:
     16. Update the best score with the new ratio
     17. Update the filename with the current file
     18. Update the image with the current fingerprint image
     19. Update kp1, kp2, and mp with the keypoints and good matches
  20. End the for loop
21. Print the best match filename and score
22. Draw the matches on the sample and fingerprint image and display the result
```

Before

```
import os
import cv2


# To read the image --> file
sample = cv2.imread("./Kaggle/images/101__M_Left_little_finger.bmp")

# How good or close is the current match vs the best match, and if so, replace the best match
best_score = 0

filename = None

# This will be the best image
image = None

# We want to have key points of the "Sample" and also "original" image
# so we can plot the connections between the individual key points.
kp1, kp2, mp = None, None, None

counter = 0

# This for loop will go through the list or directory up to 1000 images
for file in [file for file in os.listdir("./Kaggle/images")][:1000]:
    if counter % 10 == 0:
        print(counter)
        print(file)
    counter += 1

    fingerprint_image = cv2.imread("./Kaggle/images/" + file)
    sift = cv2.SIFT_create()

    keypoints_1, descriptors_1 = sift.detectAndCompute(sample, None)
    keypoints_2, descriptors_2 = sift.detectAndCompute(fingerprint_image, None)

    matches = cv2.FlannBasedMatcher({'algorithm': 1, 'trees': 10},
                                    {}).knnMatch(descriptors_1, descriptors_2, k=2)

    match_points = []

    for p, q in matches:
        if p.distance < 0.1 * q.distance:
            match_points.append(p)
    keypoints = 0
    if len(keypoints_1) < len(keypoints_2):
        keypoints = len(keypoints_1)
    else:
        keypoints = len(keypoints_2)

    if len(match_points) / keypoints * 100 > best_score:
        best_score = len(match_points) / keypoints * 100
        filename = file
        image = fingerprint_image
        kp1, kp2, mp = keypoints_1, keypoints_2, match_points

print("BEST MATCH: " + filename)
print("SCORE " + str(best_score))

result = cv2.drawMatches(sample, kp1, image, kp2, mp, None)
result = cv2.resize(result, None, fx=4, fy=4)
cv2.imshow("Result", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

After

```
import os
import cv2

# Read the sample image file
sample = cv2.imread("./Kaggle/images/101__M_Left_little_finger.bmp")

# Set initial values for best_score and filename
best_score = 0
filename = None

# Set initial values for key points
kp1, kp2, mp = None, None, None

# Set counter to 0
counter = 0

# Loop through each file in the directory up to 1000 images
for file in [file for file in os.listdir("./Kaggle/images")][:1000]:
    # Check if counter is a multiple of 10 and print the current file being processed
    if counter % 10 == 0:
        print(counter)
        print(file)

    # Increment counter by 1
    counter += 1

    # Read the fingerprint image file
    fingerprint_image = cv2.imread("./Kaggle/images/" + file)

    # Create a SIFT object
    sift = cv2.SIFT_create()

    # Detect and compute the key points and descriptors of the sample and fingerprint images
    keypoints_1, descriptors_1 = sift.detectAndCompute(sample, None)
    keypoints_2, descriptors_2 = sift.detectAndCompute(fingerprint_image, None)

    # Match the descriptors using the FLANN-based matcher
    matches = cv2.FlannBasedMatcher({'algorithm': 1, 'trees': 10}, {}).knnMatch(descriptors_1,
descriptors_2, k=2)

    # Filter the matches using the ratio test and store the good matches
    match_points = []
    for p, q in matches:
        if p.distance < 0.1 * q.distance:
            match_points.append(p)

    # Determine the number of keypoints to calculate the score
    keypoints = 0
    if len(keypoints_1) < len(keypoints_2):
        keypoints = len(keypoints_1)
    else:
        keypoints = len(keypoints_2)

    # Calculate the score
    score = len(match_points) / keypoints * 100

    # Update the best_score and filename if the current score is higher than the previous best_score
    if score > best_score:
        best_score = score
        filename = file
        image = fingerprint_image
    kp1, kp2, mp = keypoints_1, keypoints_2, match_points

# Print the best match and its score
print("BEST MATCH: " + filename)
print("SCORE " + str(best_score))

# Draw the matches and display the result
result = cv2.drawMatches(sample, kp1, image, kp2, mp, None)
result = cv2.resize(result, None, fx=4, fy=4)
cv2.imshow("Result", result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

REFERENCES

1. *Biometric Authentication - an overview | ScienceDirect Topics*. (n.d.). Biometric Authentication - an Overview | ScienceDirect Topics. https://doi.org/10.1016/B978-0-12-407189-6.00002-9

2. *Welcome to opencv documentation! — OpenCV 2.4.13.7 documentation*. (n.d.). https://docs.opencv.org/2.4/index.html

3. https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

4. Star Link Communication Pvt. Ltd. (2016, September 30). *How Fingerprint Recognition Works ? || Biometric Devices || Star Link* [Video]. YouTube. https://www.youtube.com/watch?v=AZkc48X5yck