Faceted Data

- **Do you think this is a good approach to protect systems from data leakage? What are the pros and cons?**

This approach to system protection depends on applying strict controls over data access, movement, and processing. Below is a consideration of the pros and cons of such approaches:

Pros:

1. **Prevention of Unauthorized Access**: By strictly controlling access, the risk of unauthorized data retrieval or sharing is minimized.
2. **Audit Trails**: Implementing logs and monitoring systems helps track data usage and spot anomalies, aiding in early detection of leakage.
3. **Automation**: Systems can automate data handling, reducing human errors that often lead to leaks.
4. **Encryption**: Encrypting sensitive data ensures that even if leaked, the data remains secure and unreadable.
5. **Policy Enforcement**: The system can enforce data handling policies consistently across users and systems.

Cons:

1. **Overhead**: Constant monitoring and encryption may lead to increased computational and operational overhead.
2. **False Positives**: Intrusion detection systems or monitoring tools can generate false positives, potentially disrupting legitimate activities.
3. **Complexity**: Designing a comprehensive system can be complex and may require substantial resources.
4. **Human Factor**: Users can still circumvent systems if security policies are not user-friendly or well-implemented.
5. **Insider Threats**: Protecting against internal leaks remains challenging as users with access rights can misuse data.

(Schmitz et al, 2016)

- **Creating such system in Python**

Users > george.koridze > Desktop > Essex > SSD > ePortfolio > Unit9 > Faceted_data.py > ...

```python
import os
import logging
from cryptography.fernet import Fernet
import time

# Generate and store an encryption key
def generate_key():
    key = Fernet.generate_key()
    with open("key.key", "wb") as key_file:
        key_file.write(key)

# Load the encryption key
def load_key():
    return open("key.key", "rb").read()

# Encrypt data
def encrypt_data(data):
    key = load_key()
    fernet = Fernet(key)
    return fernet.encrypt(data.encode())

# Decrypt data
def decrypt_data(encrypted_data):
    key = load_key()
    fernet = Fernet(key)
    return fernet.decrypt(encrypted_data).decode()

# Log access attempts
def log_access(user, action, status="Success"):
    logging.basicConfig(filename="access_log.txt", level=logging.INFO,
                        format="%(asctime)s - %(user)s - %(action)s - %(status)s")
    logging.info("", extra={"user": user, "action": action, "status": status})

# Mock function to simulate data access
def access_data(user, action, data):
    try:
        log_access(user, action)
        encrypted = encrypt_data(data)
        decrypted = decrypt_data(encrypted)
        return decrypted
    except Exception as e:
        log_access(user, action, status="Failed")
        raise e
```

```python

# Example usage
if __name__ == "__main__":
    generate_key()  # Generate a key (run once)

    # Simulate access
    try:
        user_data = access_data("user1", "Read", "Sensitive Information")
        print("Decrypted Data:", user_data)
    except Exception as error:
        print("Access Denied:", error)

```

References

Schmitz, M., Averbeck, M., Böhme, T. & Nickel, S., 2016. **Faceted data exploration: A framework for managing and analyzing multidimensional data**. *Journal of Information Science*, 42(3), pp. 437–450. https://doi.org/10.1177/0165551515596717