Experiment No 6

AIM: To simulate basic IoT security measures.

Objective: Test IoT security vulnerabilities using Python tools.

Tools Used: Google Colab, Python libraries (hashlib)

Theory: The Internet of Things (IoT) connects a vast number of devices, making security a crucial concern. IoT security involves protecting devices, networks, and data from unauthorized access, breaches, and cyber threats.

Key IoT security measures include:

- **Data Encryption:** Ensures that sensitive data is protected by converting it into unreadable text for unauthorized users. Cryptographic hashing algorithms such as SHA-256 provide a secure way to store passwords and verify data integrity.
- **Authentication Mechanisms:** Techniques like two-factor authentication (2FA) and device identity verification help restrict unauthorized access to IoT systems.
- **Secure Communication Protocols:** Secure protocols such as TLS and MQTT with SSL/TLS encryption safeguard data in transit.
- **Access Control:** Role-based access control (RBAC) and strong password policies limit exposure to unauthorized individuals.
- **Threat Detection and Prevention:** Monitoring systems can identify and prevent unusual activities, mitigating security risks.

Understanding hashlib in Python

hashlib is a Python module that provides cryptographic hashing functions, ensuring data integrity and security. It supports multiple hashing algorithms, including MD5, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512.

- **SHA-256**: This is a cryptographic hash function that generates a 256-bit (32-byte) hash value. It is widely used for data integrity and password storage.
- **MD5**: Though still available in hashlib, MD5 is **not recommended for security-sensitive applications** as it is vulnerable to collision attacks.
- **HMAC (Hashed Message Authentication Code)**: A method for message authentication using a cryptographic hash function combined with a secret key.

Usage of hashlib in IoT Security

- Storing hashed passwords to prevent credential leaks.
- Verifying firmware integrity before updates.
- Securing communication between IoT devices by hashing messages.

Why MD5 is Not Secure?

- **Collision Vulnerability:** Two different inputs can produce the same hash, compromising data integrity.

- **Speed:** MD5 is fast, making it vulnerable to brute-force attacks.
- **Not Recommended for Security Applications:** Avoid using MD5 for password storage, digital signatures, or secure communications.
- Instead, **SHA-256 or higher versions** (SHA-384, SHA-512) are recommended for secure hashing.

Program Code:

```
import hashlib
import random
import time

def hash_data(data):
    """Hashes input data using SHA-256 algorithm."""
    return hashlib.sha256(data.encode()).hexdigest()

def verify_hash(original_data, hashed_data):
    """Verifies whether the hash of the given data matches the stored hash."""
    return hash_data(original_data) == hashed_data

try:
    while True:
        # Generate random IoT sensor data
        temperature = round(random.uniform(20.0, 30.0), 1)
        humidity = random.randint(40, 80)
        sensor_data = f"Temperature: {temperature}°C, Humidity: {humidity}%"

        # Apply Hashing
        hashed_data = hash_data(sensor_data)
        verification = verify_hash(sensor_data, hashed_data)

        # Print results
        print("\nOriginal Data:", sensor_data)
        print("Hashed Data:", hashed_data)
        print("Verification Successful:", verification)

        # Pause before generating new data
        time.sleep(2)

except KeyboardInterrupt:
    print("\nProcess stopped by user.")
```

Explanation of LOGIC:

1. **Simulated IoT Data:** A sample sensor reading (temperature and humidity) is used to demonstrate data security concerns.
2. **Hashing Function:** The hash_data function uses SHA-256 to securely hash the sensor data.
3. **Verification Function:** Compares the hash of the original data with the stored hash to ensure integrity.

4. **Data Protection:** The hashed output ensures that even if data is intercepted, it cannot be easily reversed.
5. **Output Verification:** The program prints the original and hashed data for comparison and integrity verification.
6. **Continuous Execution:** The program runs in a loop, repeatedly generating new values and encrypting them until Ctrl+C is pressed to stop execution.

Message Flow:

1. Generate simulated IoT sensor data.
2. Apply hashing to protect data.
3. Verify data integrity.
4. Display both original and hashed data.
5. Repeat Until User Stops Execution (Ctrl+C)

Flowchart:

Start Program ↓
Generate Simulated IoT Data ↓
Apply SHA-256 Hashing ↓
Verify Hashed Data ↓
Display Secured Data ↓
Loop Until Ctrl+C ↓
End

Observation Table:

Sample	Original IoT Data	Hashed Data (SHA-256)	Verification Status
1	Temperature: 23.5°C, Humidity: 40%	3bc20d74742d7345b191f82727c4223c3c8c5c310b42c40f10916d60d925bd6a	Successful: True
2	Temperature: 21.8°C, Humidity: 48%	0f796cf2a194fef2afda461c0a38caea3aad92e411f5fa42c33d7ac318507fe3	Successful: True
3	Temperature: 22.4°C, Humidity: 59%	f422194c14f170ecefd942a16a68b461a4bbef1f336217d6225492a0504a72e6	Successful: True
4	Temperature: 22.8°C, Humidity: 41%	9a7b8c7ec11f3446f8a17ad9018d7991e9186231ece89a90c0532aeb74a8d288	Successful: True

Outcome:

```
Original Data: Temperature: 23.5°C, Humidity: 40%
Hashed Data: 3bc20d74742d7345b191f82727c4223c3c8c5c310b42c40f10916d60d925bd6a
Verification Successful: True

Original Data: Temperature: 21.8°C, Humidity: 48%
Hashed Data: 0f796cf2a194fef2afda461c0a38caea3aad92e411f5fa42c33d7ac318507fe3
Verification Successful: True

Original Data: Temperature: 22.4°C, Humidity: 59%
Hashed Data: f422194c14f170ecefd942a16a68b461a4bbef1f336217d6225492a0504a72e6
Verification Successful: True

Original Data: Temperature: 22.8°C, Humidity: 41%
Hashed Data: 9a7b8c7ec11f3446f8a17ad9018d7991e9186231ece89a90c0532aeb74a8d288
Verification Successful: True

Original Data: Temperature: 25.3°C, Humidity: 63%
Hashed Data: 70bad25b5132c06d1770a2e493fa32a0ca4e9d3722ae9d19b499b7d50beba518
Verification Successful: True

Process stopped by user.
```

Conclusion:
IoT security is essential to prevent cyber threats and unauthorized access. Using encryption techniques such as hashing helps secure transmitted and stored data. This experiment highlights the importance of applying security measures to protect IoT ecosystems.

Homework Assigned:

AIM: To implement cryptographic hashing and HMAC (Hashed Message Authentication Code) for IoT security.

Objective: Extend the IoT security program by incorporating HMAC to ensure message authenticity.

Tools Used: Google Colab, Python libraries (hashlib, hmac)

Program Code:

```
import hashlib
import hmac
import random
import time

def generate_hmac(data, key):
    """Generates an HMAC authentication code using MD5."""
    return hmac.new(key.encode(), data.encode(), hashlib.md5).hexdigest()
```

```
def verify_hmac(data, key, received_hmac):
    """Verifies if the received HMAC matches the computed one."""
    computed_hmac = generate_hmac(data, key)
    return computed_hmac == received_hmac

def hash_data(data):
    """Hashes input data using SHA-256 algorithm."""
    return hashlib.sha256(data.encode()).hexdigest()

def generate_sensor_data():
    """Generates random sensor data for Temperature, Humidity, and Pressure."""
    temperature = round(random.uniform(20.0, 30.0), 1)
    humidity = round(random.uniform(40.0, 70.0), 1)
    pressure = round(random.uniform(900.0, 1100.0), 1)
    return f"Temperature: {temperature}°C, Humidity: {humidity}%, Pressure: {pressure} hPa"

secret_key = "IoT_Secret_Key"
try:
    while True:
        sensor_data = generate_sensor_data()
        hashed_data = hash_data(sensor_data)
        hmac_data = generate_hmac(sensor_data, secret_key)

        print("\nOriginal Data:", sensor_data)
        print("Hashed Data:", hashed_data)
        print("Generated HMAC:", hmac_data)
        print("Verification Successful:", verify_hmac(sensor_data, secret_key, hmac_data))

        time.sleep(2)  # Wait for 2 seconds before generating new values
except KeyboardInterrupt:
    print("\nProcess stopped by user.")
```

Explanation of LOGIC:

1. **Simulated IoT Data**: Example sensor readings to demonstrate HMAC.
2. **Hashing Function:** The hash_data function uses SHA-256 to securely hash the sensor data.
3. **HMAC Generation**: The `generate_hmac` function creates an authentication code using MD5.
4. **Verification Function**: `verify_hmac` ensures that the received HMAC matches the expected value.
5. **Data Protection:** Unauthorized modifications will result in verification failure.
6. **Output Verification:** The program prints the original and hashed data for comparison and integrity verification.
7. **Continuous Execution:** The program runs in a loop, repeatedly generating new values and encrypting them until Ctrl+C is pressed to stop execution.

Message Flow:

1. Start the program

2. Continuously Generate Random IoT Sensor Data
3. Compute SHA-256 hash for the data.
4. Compute HMAC for the data using a secret key.
5. Verify the authenticity of the data using HMAC.
6. Display the original data and the HMAC for validation.
7. Repeat Until User Stops Execution (Ctrl+C)

Flowchart:
Start ↓
Generate Random Sensor Data ↓
Apply SHA-256 Hashing ↓
Compute HMAC with Secret Key ↓
Verify HMAC Authentication ↓
Display Results ↓
Loop Until Ctrl+C ↓
End

Observation Table:

Sample	Original IoT Data	Hashed Data (SHA-256)	Generated HMAC	Verification Status
1	Temperature: 27.7°C, Humidity: 40.7%, Pressure: 998.5 hPa	ab010cdc41516885811fb8fc905796dd52c8425cfad74938ef4ff8192d04dd6f	0417d1832f1742c3fb637e9fb46353d2	Successful: True
2	Temperature: 20.3°C, Humidity: 63.4%, Pressure: 939.1 hPa	0eb6ae15c2b327e0ca137b0bc64619292746f0f522efaf1551fb10a8d0cfb4fd	2a44013709d40a05663a9645ecbf5af3	Successful: True
3	Temperature: 20.9°C, Humidity: 57.5%, Pressure: 940.0 hPa	59da2746aceb1948ca0c9e65c4079e4f48deba4e7f289674e984e7d996f6036c	5095f501eab7064d2f0e09b1fbdebbd2	Successful: True
4	Temperature: 26.9°C, Humidity: 42.4%, Pressure: 1093.6 hPa	8297427784c316703debe5889e42f0dae44cf7e47632e724c16343b2f09fbfa8	7c77178e6c2e9a156e2ae7813b21bcf4	Successful: True

Outcome:

```
Original Data: Temperature: 27.7°C, Humidity: 40.7%, Pressure: 998.5 hPa
Hashed Data: ab010cdc41516885811fb8fc905796dd52c8425cfad74938ef4ff8192d04dd6f
Generated HMAC: 0417d1832f1742c3fb637e9fb46353d2
Verification Successful: True

Original Data: Temperature: 20.3°C, Humidity: 63.4%, Pressure: 939.1 hPa
Hashed Data: 0eb6ae15c2b327e0ca137b0bc64619292746f0f522efaf1551fb10a8d0cfb4fd
Generated HMAC: 2a44013709d40a05663a9645ecbf5af3
Verification Successful: True

Original Data: Temperature: 20.9°C, Humidity: 57.5%, Pressure: 940.0 hPa
Hashed Data: 59da2746aceb1948ca0c9e65c4079e4f48deba4e7f289674e984e7d996f6036c
Generated HMAC: 5095f501eab7064d2f0e09b1fbdebbd2
Verification Successful: True

Original Data: Temperature: 26.9°C, Humidity: 42.4%, Pressure: 1093.6 hPa
Hashed Data: 8297427784c316703debe5889e42f0dae44cf7e47632e724c16343b2f09fbfa8
Generated HMAC: 7c77178e6c2e9a156e2ae7813b21bcf4
Verification Successful: True

Original Data: Temperature: 22.0°C, Humidity: 57.2%, Pressure: 1034.2 hPa
Hashed Data: 1f4d09530f9838bc0e0bbb7b2e3d3d15d2c2040ba30eda4a39bdbee765bcd53c
Generated HMAC: a4d990e090fa180dc5c46be01c824bc5
Verification Successful: True

Process stopped by user.
```

Conclusion:

HMAC provides a reliable way to ensure data authenticity and integrity in IoT environments. By combining cryptographic hashing with a secret key, it protects against tampering and unauthorized access. This experiment highlights the importance of secure authentication techniques in IoT security.