Experiment 6

AIM: To simulate basic IoT security measures.

Objective: Test IoT security vulnerabilities using Python tools.

Tools Used: Google Colab, Python libraries (hashlib)

Theory: 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:** Random temperature and humidity values are generated as sensor data.
2. **Hashing Function:** The `hash_data()` function converts sensor data into a **SHA-256 hash** for security.
3. **Verification Function:** The `verify_hash()` function checks if the original data's hash matches the stored hash.
4. **Data Protection:** The hash ensures data cannot be **tampered with or reversed**.
5. **Output Verification:** The program displays the original data, its hash, and the **verification status**.

Message Flow:

1. Generate random IoT sensor data.
2. Apply **SHA-256** hashing.
3. Verify hash integrity.
4. Print results.
5. Repeat every **2 seconds** until stopped.

Flowchart:

Start Program ↓
Generate Simulated IoT Data ↓
Apply SHA-256 Hashing ↓
Verify Hashed Data ↓
Display Secured Data ↓
End

Observation Table:

Sample	Original IoT Data	Hashed Data (SHA-256)	Verification Status
1	Temperature: 29.2°C, Humidity: 52%	066b35be3f4902e234cc336590f1468558d89d970219619411c79262dbf92618	Successful
2	Temperature: 21.2°C, Humidity: 57%	295124085a36a006780353e33fefb0aadca1cd5631877394399507ae6d4858a1	Successful
3	Temperature: 22.2°C, Humidity: 66%	cfc0ceeb85e4ce9b3dd10bdb23d375aa976286bc902c131d48a5901f55458c83	Successful

4	Temperature: 23.7°C, Humidity: 60%	9bebb84417a3f6b67931eb9b1 1450712060db4bab988877e2 42002ce3261a216	Successful

Outcome:

```
Original Data: Temperature: 29.2°C, Humidity: 52%
Hashed Data: 066b35be3f4902e234cc336590f1468558d89d970219619411c79262dbf92618
Verification Successful: True

Original Data: Temperature: 21.2°C, Humidity: 57%
Hashed Data: 295124085a36a006780353e33fefb0aadca1cd5631877394399507ae6d4858a1
Verification Successful: True

Original Data: Temperature: 22.2°C, Humidity: 66%
Hashed Data: cfc0ceeb85e4ce9b3dd10bdb23d375aa976286bc902c131d48a5901f55458c83
Verification Successful: True

Original Data: Temperature: 23.7°C, Humidity: 60%
Hashed Data: 9bebb84417a3f6b67931eb9b11450712060db4bab988877e242002ce3261a216
Verification Successful: True

Original Data: Temperature: 21.2°C, Humidity: 57%
Hashed Data: 295124085a36a006780353e33fefb0aadca1cd5631877394399507ae6d4858a1
Verification Successful: True

Original Data: Temperature: 26.8°C, Humidity: 55%
Hashed Data: 38f68e97271090a567d2d3835646c90e8f57cdacfbeccaf3bea3edfa8da100b1
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 simulate basic IoT security measures using HMAC.

Objective: Test IoT security vulnerabilities using Python tools with HMAC for message authentication.

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

Theory: The Internet of Things (IoT) connects numerous devices, making security a significant concern. IoT security involves protecting data and devices from unauthorized access. HMAC

(Hashed Message Authentication Code) combines cryptographic hash functions with a secret key to ensure both data integrity and authenticity.

Key IoT security measures with HMAC include:

- Data Integrity: Ensures that the message is not tampered with during transmission.
- Authentication: Verifies that the message is from a trusted source.
- Secret Key: Provides an extra layer of security compared to simple hashing.

Understanding hmac in Python: The `hmac` library in Python helps generate a hashed message authentication code using cryptographic hash functions like MD5, SHA-1, or SHA-256.

Program Control :

```
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:

- Simulated IoT Data: Random temperature and humidity values are generated.
- HMAC Function: Uses the hmac library to generate an authentication code with MD5 and a secret key.
- Data Protection: Ensures both the integrity and authenticity of the data.
- Output Verification: Displays the original data and HMAC code.
- Repetition: Continuously generates new data until stopped by the user.

Message Flow:

1. Generate simulated IoT sensor data.
2. Apply HMAC with a secret key.
3. Display original data and HMAC.
4. Repeat the process.

Flowchart:

Start Program ↓

Generate Simulated IoT Data ↓

Apply HMAC ↓

Display Secured Data ↓

End

Observation Table:

Sr. No	Original IOT Data	Hashed Data	Generated HMAC	Verification

1	Temperature: 21.2°C, Humidity: 51.9%, Pressure: 939.2 hPa	0dfd43e6b70eeaed09ad0600a4bdf5f8d6e39c29a11dffc7ff31103b3debc318	3a99f39a0c18c5e7f603e04371116f95	Successful
2	Temperature: 27.1°C, Humidity: 44.5%, Pressure: 1084.2 hPa	ccb7086ab83ea149cd23b9703a8c6f6410a9cc865275a10b3e18b511e6a00448	42bc666922263140ad92de495efc6146	Successful
3	Temperature: 21.3°C, Humidity: 61.5%, Pressure: 1045.5 hPa	7b837145272b2e0be2495bebdf474588033d96a73e5bfcae25712c467c43843e	7b9a5bb711181c993ec17c6e5313e9e4	Successful
4	Temperature: 28.4°C, Humidity: 59.0%, Pressure: 1079.7 hPa	0b48f7adb5d5e16e89525ca4dd51771cef214002227ce44c00533eab906eac92	c25bc313b3c1d4886d2ce83f9bbabfea	Successful

Outcome :

```
Original Data: Temperature: 21.2°C, Humidity: 51.9%, Pressure: 939.2 hPa
Hashed Data: 0dfd43e6b70eeaed09ad0600a4bdf5f8d6e39c29a11dffc7ff31103b3debc318
Generated HMAC: 3a99f39a0c18c5e7f603e04371116f95
Verification Successful: True

Original Data: Temperature: 27.1°C, Humidity: 44.5%, Pressure: 1084.2 hPa
Hashed Data: ccb7086ab83ea149cd23b9703a8c6f6410a9cc865275a10b3e18b511e6a00448
Generated HMAC: 42bc666922263140ad92de495efc6146
Verification Successful: True

Original Data: Temperature: 21.3°C, Humidity: 61.5%, Pressure: 1045.5 hPa
Hashed Data: 7b837145272b2e0be2495bebdf474588033d96a73e5bfcae25712c467c43843e
Generated HMAC: 7b9a5bb711181c993ec17c6e5313e9e4
Verification Successful: True

Original Data: Temperature: 28.4°C, Humidity: 59.0%, Pressure: 1079.7 hPa
Hashed Data: 0b48f7adb5d5e16e89525ca4dd51771cef214002227ce44c00533eab906eac92
Generated HMAC: c25bc313b3c1d4886d2ce83f9bbabfea
Verification Successful: True

Original Data: Temperature: 26.8°C, Humidity: 55.9%, Pressure: 1020.4 hPa
Hashed Data: 3661c99c33ee9d90d40b0b11725af38540cfe82009d0f0dc94bd3daba4a89b06
Generated HMAC: 8387ab9ace4fab75e22cee452a51e692
Verification Successful: True

Process stopped by user.
```

Conclusion: Implementing HMAC helps secure IoT data by verifying its authenticity and integrity. This experiment highlights how secret keys and cryptographic hashing enhance IoT security.