University of Sheffield

COM1001 SPRING SEMESTER

Professor Phil McMinn

p.mcminn@sheffield.ac.uk

Data Confidentiality

What is “Data Confidentiality”?

Confidentiality means
ensuring data is only accessible
to those authorised to view it.

Examples of sensitive data you might want kept confidential include **addresses**, **phone numbers, bank account numbers and passwords.**

Why Must We Take It Seriously?

As the maintainer of a web service, it’s important that you keep sensitive user data confidential

Is you suffer a confidentiality breach, and did not follow proper practices, you may be liable for damages!

https://haveibeenpwned.com

How Can We Keep Data Confidential?

First line:

- Sanitise external inputs
- Prevent unauthorised database acces, e.g. via SQL injection (Sequel handles for us)

Second line:

- **Encrypt sensitive data**, making it unreadable without knowing the user's password

How to Encrypt User Data

Use an established software library such as OpenSSL

There are two important algorithms you'll need from this library to secure the user data in your projects:

- **Advanced Encryption Standard (AES)** to perform the actual encryption and decryption
- **Password-Based Key Derivation Function 2 (PBKDF2)** to convert a user's password into a suitable **key** (a fixed length string)

Advanced Encryption Standard (AES)

AES is a “symmetric” encryption algorithm: this means it **encrypts and decrypts data using the same key**. **The key must be kept secret!**

Requires an **“initialisation vector” (IV)**, which sets the initial state of the algorithm. **This does not need to be secret, but should be random.**

A different IV should be used for each user. This means that should two users have the same key and the same data, their encrypted data will be different. (While unlikely, this is still good practice.)

Password-Based Key Derivation Function 2 (PBKDF2)

Converts a user's password into a key for AES.

AES requires a fixed length key (e.g. 16 bytes), so using a password directly is not usually possible.

Requires a salt – a random bit string. **This must be different for every user.** PBKDF2 combines the password and salt to ensure that users with the same password get different keys (mitigating against rainbow table attacks).

PBKDF2 does not make a password harder to guess. It should not be considered a substitute for a strong password.

Procedure for Encrypting User Data

1. Generate a random IV and salt for the user
2. Use PBKDF2 with the user's password and salt to produce the key
3. Use AES with the IV and key to encrypt the data
4. Store the user's IV, salt and encrypted data in the database

```
# Set up AES for encryption.
aes = OpenSSL::Cipher.new('AES-128-CBC').encrypt

# Generate a random IV.
self.iv = Sequel.blob(aes.random_iv)

# Generate a random salt.
self.salt = Sequel.blob(OpenSSL::Random.random_bytes(16))

# Get the key from the password and salt.
aes.key = OpenSSL::PKCS5.pbkdf2_hmac_sha1(password, salt, 20_000,
16)
# Encrypt the data
elf.data_crypt = Sequel.blob(aes.update(data) + aes.final)
```

Procedure for Decrypting User Data

1. Read the user's IV, salt, and encrypted data from the database

2. Use PBKDF2 and the user's password and salt to produce the key.

3. User AES with the IV and key to decrypt the data

```
# Set up AES for decryption.
aes = OpenSSL::Cipher.new('AES-128-CBC').decrypt

# Set the IV.
aes.iv = iv

# Get the key from the password and salt.
aes.key = OpenSSL::PKCS5.pbkdf2_hmac_sha1(password, salt, 20_000,
16)
# Try to decrypt the data.
begin
  return aes.update(data_crypt) + aes.final
rescue OpenSSL::Cipher::CipherError
  return nil
end
```

Live Demonstration:

`week3/data_confidentiality_example`

(from the COM1001 GitHub repository)

Featuring:

- The database structure
- Demonstrating signup – and the encryption int the database
- Logging in following signup