

Encryption Testing

Encryption testing involves verifying the security and correctness of the encryption and decryption processes in a system. The goal is to ensure that sensitive data is protected against unauthorized access and that the encryption implementation does not have vulnerabilities that could be exploited to reveal the data.

Creating Encryption Testing

When conducting encryption testing, you typically:

- Check the strength and appropriateness of the encryption algorithms and key sizes used.
- Verify that cryptographic keys are stored and managed securely.
- Ensure that encryption and decryption processes are correctly implemented without introducing weaknesses.
- Confirm that error handling does not leak sensitive information.
- Evaluate the randomness and unpredictability of initialization vectors (IVs) and salts.

What Will We Do?

In this lesson, we will use [prompt engineering](#) to aid in creating encryption tests. We will be supplied with Python [code](#) containing no encryption unit tests. We must prompt the AI to create some tests to ensure that our code is free from security vulnerabilities. (*Note: You can copy the code into the chat to help get better results.*)

Instructions

Below is a Python script with known encryption vulnerabilities.

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend

key = b'insecurekey'
iv = b'insecureiv'

def encrypt(data):
    # This function encrypts data with a hardcoded key and IV.
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    return encryptor.update(data) + encryptor.finalize()

def decrypt(encrypted_data):
    # This function decrypts data with a hardcoded key and IV.
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    return decryptor.update(encrypted_data) + decryptor.finalize()
```

Your task is to prompt the AI chat to develop unit tests for the Python script. **You will know you have successfully completed the task when you obtain the following results from the machine:**

- Using the Python unittest framework
- Using the Python encrypt and decrypt libraries
- A test that check if the key length is at least 128 bits (16 bytes)
- A test that check if the IV length is at least 128 bits (16 bytes)
- A test that ensures the encrypt and decrypt functions work correctly
- A test to ensure that the IV is random and unpredictable for each encryption.

Note: Depending on your prompt, you may receive different results than anticipated. That is okay. However, there should be at least four (4) tests.

If you would like some help creating prompts to send to the AI, open the hint dropdown below.

Prompt Hint

The AI remembers previous messages so you can chat back and forth like a conversation. You may need to use multiple prompts, but combining prompts saves computation time and energy. Here are some examples below:

- Create unit tests for the following Python code using the unittest, encrypt, and decrypt libraries: <paste in the code from above>
- Add tests to check if the key length and IV length are at least 128 bits, tests to ensure the encrypt and decrypt functions work correctly, and tests to ensure that the IV is random and unpredictable for each encryption.

If you need the AI to forget past prompts, click the trash can icon and the chat will reset.

Prompt

Create unit tests for the following Python code using the unittest, encrypt, and decrypt libraries: f

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
```

```
key = b'insecurekey'
iv = b'insecureiv'
```

```
def encrypt(data):
    # This function encrypts data with a hardcoded key and IV.
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    return encryptor.update(data) + encryptor.finalize()
```

```
def decrypt(encrypted_data):
    # This function decrypts data with a hardcoded key and IV.
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
```

```
decryptor = cipher.decryptor()
return decryptor.update(encrypted_data) + decryptor.finalize()
```

Add tests to check if the key length and IV length are at least 128 bits, tests to ensure the encrypt and decrypt functions work correctly, and tests to ensure that the IV is random and unpredictable for each encryption.

Here is an example of how you could write unit tests for the provided Python code:

```
import unittest

import os

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend

key = b'insecurekey'
iv = b'insecureiv'

def encrypt(data):
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    return encryptor.update(data) + encryptor.finalize()

def decrypt(encrypted_data):
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    return decryptor.update(encrypted_data) + decryptor.finalize()

class TestEncryption(unittest.TestCase):

    def test_key_length(self):
        self.assertGreaterEqual(len(key), 16, "Key length should be at least 128 bits (16 bytes)")

    def test_iv_length(self):
        self.assertEqual(len(iv), 16, "IV length should be 128 bits (16 bytes)")
```

```
def test_encrypt_decrypt(self):
    plaintext = b'Hello, World!'
    ciphertext = encrypt(plaintext)
    decrypted_text = decrypt(ciphertext)

    self.assertEqual(plaintext, decrypted_text, "Encryption and decryption did not produce the same
plaintext")

def test_random_iv(self):
    iv1 = os.urandom(16)
    iv2 = os.urandom(16)

    self.assertNotEqual(iv1, iv2, "IVs should be different for each encryption")

if __name__ == '__main__':
    unittest.main()
```

In the `test_random_iv` function, `os.urandom(16)` is used to generate a random IV of 16 bytes (128 bits) for testing purposes. This ensures that the IV is random and different for each encryption. After running these tests, you should be able to verify the key and IV length, ensure the encrypt and decrypt functions work correctly, and confirm that the IV is random and unpredictable for each encryption.