Team:

Ahmed Mahmoud Hassan - 2205155

Ali Mohamed Oqab - 2205077

Rewan Ahmed Elwardany - 2205218

Mitigation, Defense & Attack Demonstration

Length Extension Attack Demonstration

- Intercepting a valid (MAC , Message) pair

```
Attempting attack with key length guess: 14 bytes
Original message: amount=100&to=alice
Original MAC: 614d28d808af46d3702fe35fae67267c
```

- The server uses the vulnerable MD5 hashing formula

```python
def generate_mac(message: bytes) -> str:
    return hashlib.md5(SECRET_KEY + message).hexdigest()
```

Performing Length Extension Attack

- Attempts the length extension attack

```
new_mac, new_message = hashpumpy.hashpump(
    intercepted_mac,
    intercepted_message,
    data_to_append,
    14  # Correct length for 'supersecretkey'
)
```

Which takes the message and append the new message the attacker want to insert and then creates the new forged MAC

Then the attacker forges the MAC and the Message which will be inherited into the vulnerable server

```
Forged message: amount%3D100%26to%3Dalice%C2%80%00%00%00%0
Forged MAC: 97312a73075b6e1589117ce55e0a3ca6
```

Then for the test we insert them into server.py to test the attack

```
forged_message = b'amount=100&to=alice\x80\x00\x00\x00\x00\x00\x00\x00\x00\x0

forged_mac = "97312a73075b6e1589117ce55e0a3ca6"
```

When we run the server it will run normally and we made a mitigation to tell us if the forged message is verified successfully or not

```
----Verifying legitimate message----
MAC verified successfully. Message is authentic.

----Verifying forged message----
Forged message: amount=100&to=alic&admin=true
Forged MAC: 97312a73075b6e1589117ce55e0a3ca6
MAC verified successfully (UNEXPECTED - VULNERABLE)
```

But if we didn't insert the forged (MAC, Message) It will return false since it hasn't been altered

```
----Verifying legitimate message----
MAC verified successfully. Message is authentic.

----Verifying forged message----
Forged message: amount=100&to=alice&admin=true
Forged MAC: 614d28d808af46d3702fe35fae67267c
MAC verification failed (EXPECTED - SECURE)
```

```
if verify(forged_message, forged_mac):
    print("MAC verified successfully (UNEXPECTED - VULNERABLE)")
else:
    print("MAC verification failed (EXPECTED - SECURE)")
```

Defense Strategies

- By Using HMAC (Hash-based Message Authentication Code)

- HMAC is a construction used for message authentication, which combines a cryptographic hash function (MD5, SHA-256) with a secret key

- The goal of HMAC is to verify both the integrity and the authenticity of a message

- HMAC uses a nested structure that prevents recovery of the hash state, even if the attacker knows the MAC

Implementation in the Secure-Server:

```
import hmac
```

```
def generate_secure_mac(message: bytes) -> str:
    return hmac.new(SECRET_KEY, message, hashlib.sha256).hexdigest()
```

Why HMAC Mitigates Length Extension Attack?

1. Structure of HMAC

HMAC avoids the insecure pattern **`H(secret || message)`**

It uses a two-step hash with key mixing and padding:

HMAC(K, m) = H((K ⊕ opad) || H((K ⊕ ipad) || m))

Components

- **`K`:** Secret key
- **`m`:** Message
- **`H`:** Hash function (MD5)
- **`||`:** Concatenation
- **⊕:** XOR (bitwise)
- **`ipad`:** 0x36 repeated (Inner Padding for Mixing the secret key into the first hash operation)
- **`opad`:** 0x5C repeated (Outer Padding for Mixes the secret key into the second hash operation)

Why it's secure

- The key is never exposed directly
- The inner and outer layers each mix the key differently
- The output of the first hash is protected by the second

2. Preventing Internal State Reuse

Length extension attacks depend on reusing the internal state of the hash function.

HMAC blocks this with two techniques:

- **Inner hash isolation**
  The attacker can't continue from **`H(K ⊕ ipad || m)`** because the outer hash adds a different key layer.

- **No exposed intermediate state**
  `K ⊕ opad` wraps the inner hash. The result looks random. The attacker can't use it to forge a valid MAC.

3. Role of Padding in HMAC

The padding values serve two purposes:

- **Key masking**
  `K ⊕ ipad` and **`K ⊕ opad`** hide the original key from both hash inputs.
  Even if the MAC leaks, the key structure doesn't.

- **Randomizing the input**
  Fixed byte values ensure input unpredictability.
  This prevents any meaningful structure an attacker could exploit.

```
Attempting attack with key length guess: 14 bytes
Original message: amount=100&to=alice
Original MAC: a86f897948d15c923c1f77133e805c707ca4fa752e3960efde47d618425027d5

Forged message: amount%3D100%26to%3Dalice%C2%80%00%00%00%00%00%00%00%00%00%0
Forged MAC: af288e54948b13cc78d1d69da59ae747ab1da5ec3895df9d6e6e38618e2bcbb5

Attack failed (unexpected)
```

Summary

HMAC:

- Hides the key
- Breaks predictable patterns
- Stops length extension attacks

This is why HMAC is the standard for secure message authentication