Mitigation and Defense Against Length Extension Attacks

Introduction

The length extension attack exploits vulnerabilities in hash functions like MD5 when used in naive message authentication code (MAC) constructions, such as `hash(secret_key + message)`. The vulnerable `server.py` uses this construction, allowing attackers to append data (e.g., `&admin=true`) and compute a valid MAC without the secret key. To mitigate this, we implement HMAC in `server_hmac.py`, a secure MAC construction. This document details the modification, demonstrates the attack's failure, and explains HMAC's effectiveness.

Modifying the System to Use HMAC

The vulnerable `server.py` uses `hashlib.md5(SECRET_KEY + message)`, exposing MD5's internal state. We replace this with HMAC in `server_hmac.py`, defined as:

$$\text{HMAC}(K, m) = H((K \oplus \text{opad}) \parallel H((K \oplus \text{ipad}) \parallel m))$$

where H is the hash function (MD5), K is the secret key, and opad, ipad are constants. The modified code is:

```python
import hmac
import hashlib

SECRET_KEY = b'supersecretkey'

def generate_mac(message: bytes) -> str:
    return hmac.new(SECRET_KEY, message, hashlib.md5).hexdigest()

def verify(message: bytes, mac: str) -> bool:
    return mac == generate_mac(message)
```

Demonstrating Attack Failure

The `client.py` script tests the attack against both servers. It intercepts a message (`amount=100&to=alice`), uses `hashpumpy` to forge a new message with

&admin=true, and verifies the MAC. Against server.py, the attack succeeds, producing a valid MAC. Against server_hmac.py, it fails, as the forged MAC does not match the HMAC for the forged message. Output for server_hmac.py shows:

```
=== Attack on Secure HMAC Server ===
Original message: amount=100&to=alice
Original MAC: <hmac_mac_value>
Forged message (bytes): b'amount=100&to=alice\x80\x00...&admin=true'
Forged MAC: <new_mac_value>
--- Verifying forged message (HMAC server) ---
MAC verification failed (attack failed).
```

Why HMAC Mitigates Length Extension Attacks

Length extension attacks exploit the Merkle-Damgård construction of MD5, where the hash output is the final state, allowing attackers to append data and continue hashing. HMAC prevents this through:

- **Key Protection**: The key is XORed with ipad and opad, preventing direct concatenation.
- **Nested Hashing**: The outer hash obscures the inner hash's state.
- **Fixed-Length Input**: The inner hash's output is fixed-length, breaking extensibility.
- **Padding Isolation**: ipad and opad isolate the message's influence.

In server_hmac.py, hmac.new ensures the MAC cannot be extended, as hashpumpy cannot account for the key or HMAC's structure.

Conclusion

HMAC in server_hmac.py eliminates the length extension vulnerability. The client.py script confirms the attack's failure, as the forged MAC is rejected. HMAC's key protection, nested hashing, and isolation ensure attackers cannot extend the hash state, providing robust defense. While SHA-256 is preferred for modern systems, HMAC-MD5 effectively demonstrates mitigation.