Background write_up

a. **What is a MAC and its purpose in data integrity/authentication?**

A **Message Authentication Code (MAC)** is a cryptographic checksum generated using a secret key and a message. Its primary function is to ensure **data integrity** and **authenticity** during communication between parties who share a secret key. When a sender transmits a message, they also send a MAC. The receiver, who knows the same secret key, recalculates the MAC and compares it to the received one. If they match, the message is considered untampered and authentic.

MACs are used in many applications, such as secure communication protocols (SSL/TLS), financial transactions, and data storage verification. They help ensure that a message was not altered during transit and was indeed created by a legitimate source.

b. **How does a Length Extension Attack work in hash functions like MD5/SHA1?**

Length Extension Attacks exploit the **internal structure** of certain hash functions (e.g., **MD5**, **SHA-1**) which are based on the **Merkle–Damgård construction**. These functions process data in fixed-size blocks and maintain an internal state across iterations.

In a vulnerable construction like MAC = hash(secret || message), an attacker who knows:

- The output hash (MAC),
- The original message length,
- The hash algorithm used (e.g., MD5),

...can **append extra data** to the original message and compute a new hash **without knowing the secret key**.

Here's how the attack works:

1. The attacker uses tools like hash_extender or hashpumpy to recreate the hash state after hashing secret || message.
2. They append additional data to the message (e.g., &admin=true).
3. Using the predicted padding (according to the hash function), they compute a valid hash for the extended message.
4. The resulting forged message and hash pass server verification, despite the attacker not knowing the secret.

c. **Why is MAC = hash(secret || message) insecure?**

This construction is insecure **because it enables length extension attacks** due to how hash functions like MD5 or SHA-1 process input. These hash functions work in a block-wise fashion, and their final output depends on the internal state that can be continued if known.

By hashing secret || message, an attacker who observes the final hash output can essentially **continue the hashing process** with new data (after proper padding) and generate a valid hash for secret || message || padding || new_data.

The vulnerability lies in the **predictability** of the hash state and padding mechanism. Since the secret is prepended (not postpended), the attacker can manipulate the message and generate a valid MAC without ever knowing the key.