Length Extension Attack Project - Write-up

Mitigation Project: Length Extension Attack

1. Background Write-up

A Message Authentication Code (MAC) provides message integrity and authentication.
A naive implementation of a MAC might simply hash the concatenation of a secret key and message,
such as: MAC = Hash(secret_key || message).

This approach is insecure when used with Merkle-Damgård-based hash functions such as MD5 or SHA-1.
These hash functions are vulnerable to a class of attacks known as "length extension attacks".

Length extension attacks allow an adversary to extend the original message (even without knowing the key) and compute a valid MAC for the extended message. This is possible because the internal state of the hash function after hashing `secret || message` can be reused to hash additional data.

For instance, if an attacker knows the hash (MAC) of "secret || message", they can guess the key length and append more data to the original message, resulting in a valid MAC for "secret || message || padding || extension".

This vulnerability demonstrates why simple concatenation is not cryptographically sound for MACs.

Instead, the HMAC construction must be used, which ensures security even if the underlying hash function has
length extension properties.

HMAC uses an inner and outer hashing mechanism with distinct key pads to securely encapsulate the key, making it immune to length extension.

Always use HMAC rather than manually combining secrets and messages with standard hash functions.

Length Extension Attack Project - Write-up

2. Mitigation Write-up

The vulnerability exploited in a length extension attack stems from an insecure MAC construction:

MAC = Hash(secret || message)

To securely defend against such attacks, the HMAC (Hash-based Message Authentication Code) construction
should be used. HMAC wraps the hashing process in two layers of keyed hashing with distinct padding to neutralize the impact of internal state exposure.

HMAC computes a MAC as:

HMAC(K, M) = H((K ^ opad) || H((K ^ ipad) || M))

Where:
- K is the secret key (padded if necessary)
- M is the message
- ipad and opad are fixed inner and outer padding constants
- H is a cryptographic hash function (e.g., MD5, SHA-256)

This structure makes it cryptographically infeasible for an attacker to craft a valid MAC for any modified or extended message, even if they know the original MAC.

Python's `hmac` module provides a secure implementation out-of-the-box.

In testing, the insecure server was successfully attacked via a length extension technique. However, once the implementation was switched to use HMAC, all forgery attempts failed unless the correct secret key was known.

Recommendation: Always use standard HMAC implementations with trusted cryptographic libraries instead

of manual hashing of key-message combinations.