Preventing MAC Forgery Using HMAC

Introduction

MACs ensure data integrity and authenticity, but insecure constructions like MAC = hash(secret || message) are vulnerable. HMAC prevents such attacks, including length extension.

Why hash(secret || message) Is Insecure

Hash functions like MD5 and SHA-1 are vulnerable because of how they are designed:

- They process input in fixed-size blocks (e.g., 512 bits for MD5).
- They automatically add padding at the end of the input.
- They use an internal state that can be resumed if the input length is guessed.

So what can an attacker do?

If the attacker intercepts: A valid message like: amount=100&to=alice, And its MAC: hash(secret || message)

Then, without knowing the secret key, they can:

1. Guess the length of the key (e.g., 14 bytes).
2. Use tools like `hashpumpy` or a custom MD5 class to:

    a. Reconstruct the internal state of the hash from the known MAC.
    b. Append extra data like `&admin=true`.
    c. Generate a valid MAC for the forged message.

Example: Simulated insecure server:

- secret = b'supersecretkey'
- message = b'amount=100&to=alice'
- mac = hashlib.md5(secret + message).hexdigest()

Attacker builds:

- forged_message = b'amount=100&to=alice...[padding]...&admin=true'
- forged_mac = new_mac_generated_by_pymd5_or_hashpumpy

The vulnerable server accepts the forged message and MAC:

  - verify(forged_message, forged_mac) → True

This confirms that the system is vulnerable to a length extension attack.

Mitigation: Using HMAC

- import hmac, hashlib
- mac = hmac.new(secret, message, hashlib.md5).hexdigest()

HMAC stands for **Keyed-Hash Message Authentication Code** and is defined as:

- $HMAC(K, m) = H((K \oplus opad) \,||\, H((K \oplus ipad) \,||\, m))$

Why HMAC Is Secure

1. Double Hashing with Key Mixing

- The key is mixed with constants (ipad, opad) and used both inside and outside the hash.
- Attackers can't extend the hash because it's nested and key-dependent.

2. Internal State Isolation

- The internal state is hidden, so attackers can't extend the hash even if they know the HMAC output.
- HMAC prevents all padding and continuation tricks used in length extension attacks.

3. Example: Secure HMAC Construction

- secret = b'supersecretkey'
- message = b'amount=100&to=alice'
- mac = hmac.new(secret, message, hashlib.md5).hexdigest()

If an attacker tries to forge a message using tools like pymd5 or hashpumpy, the result is:

- verify(forged_message, forged_mac) → False

Result of the Mitigation

After switching to HMAC: Forged messages fail verification, The server detects tampering, The system is now resistant to length extension attacks.

References

- RFC 2104 – HMAC Spec / OWASP – Message Authentication Code (MAC) / Crypto StackExchange – Why HMAC Prevents Length Extension