Data Integrity and Authentication

Mitigation Write-Up: Securing MACs with HMAC

Submitted by:

Mohammed Abdulrahman Awad Khaled (ID: 2205114)

Mohammed Ahmed Ramadan Al-Arjawi (ID: 2205043)

Omar Ahmed Hameed Mohammed (ID: 2205213)

Submitted to:

Dr. Maged Abdelaty

May , 2025

1 Introduction

The length extension attack exploits the insecure MAC construction MAC = MD5($secret||message$), allowing attackers to forge valid MACs for extended messages without the secret key. To address this vulnerability, we implemented a secure system using HMAC (Hash-based Message Authentication Code), which is resistant to such attacks. This write-up details the HMAC implementation, explains its effectiveness, and offers recommendations for modern cryptographic practices.

2 HMAC Implementation

HMAC is defined as $\text{HMAC}(K, m) = \text{hash}((K \oplus \text{opad})||\text{hash}((K \oplus \text{ipad})||m))$, where:

- K: Secret key.
- opad, ipad: Constant padding values (outer and inner).
- hash: Cryptographic hash function (MD5 in this case).

In our implementation (`secure_server.py`):

- We utilize Python's `hmac` module with `hashlib.md5`.
- The `generate_hmac` function computes the HMAC for a given message.
- The `verify_hmac` function employs `hmac.compare_digest` for secure, constant-time comparison to mitigate timing attacks.

The implementation tests the forged message-MAC pair from the attack, which is rejected, confirming HMAC's security.

3 Why HMAC Prevents the Attack

The length extension attack relies on the exposed internal state of MD5($secret||message$), enabling attackers to append data and continue hashing. HMAC prevents this through:

- Dual hash operations: The inner hash $\text{hash}((K \oplus \text{ipad})||m)$ is hashed again with $K \oplus \text{opad}$.
- State protection: The outer hash obscures the inner hash's state, preventing extension without the key.
- Key dependency: Modifications to the message invalidate the HMAC, as the attacker cannot compute valid inner or outer hashes.

Our demonstration shows the HMAC-based server rejecting the forged message (`amount=100&to=alice` with padding and `&admin=true`) and its MAC (`ec4488b7e7bd24418b8ab38b6e5ae927`), as the attacker lacks the secret key.

4 Demonstration Results

Executing `secure_server.py` yields:

```
=== Secure Server Simulation ===
Original message: amount=100&to=alice
HMAC: 616843154afc11960423deb0795b1e68

--- Verifying legitimate message ---
 HMAC verified successfully. Message is authentic.

--- Verifying forged message with HMAC ---
 Forgery detected and rejected!
```

This confirms that HMAC effectively protects the system's integrity and authenticity.

5 Recommendations

Although HMAC-MD5 is secure against length extension attacks, MD5's collision vulnerabilities make it outdated. We recommend:

- Adopting HMAC with SHA-256 or SHA-3 for enhanced security.
- Regularly updating cryptographic libraries to address emerging threats.
- Using standard libraries like Python's `hmac` to avoid implementation errors.
- Incorporating defenses like nonces or timestamps to prevent replay attacks.

By implementing HMAC and adhering to best practices, systems can achieve robust data integrity and authentication.