Message Authentication Codes and Length Extension Attacks

1. Background Study

Name :

Ahmed Mohammed Adel Ibrahim - **2205096**
Abdelrahman Ayman Saad Abdelhalim - **2205033**
Ahmed Mohammed Bekhit - **2205136**

COURSE : Data Integrity and Authentication

Doctor : Maged Abdelaty

References : RFC 2104-Cryptanalysis of MD5 and SHA-1- Glenn Askins

1. Introduction : What is Message Authentication Codes :

Message Authentication Codes provide **symmetric-key data integrity and authenticity**. Given a secret key K and message M, a MAC algorithm outputs a fixed-size tag T such that only parties knowing K can generate or verify T:

T = MAC(K, M)

Upon receipt, the verifier recomputes MAC(K, M) and compares it to T. A match ensures:

- **Integrity**: M was not altered in transit.
- **Authenticity**: M originates from an entity possessing K.

MACs are widely used in network protocols, APIs, and software packages to protect messages and files.

---

2. MAC = hash(K || M): Naive Construction :

A simple MAC approach prepends the key to the message:

NaiveMAC(K, M) = H(K || M)

where H is a cryptographic hash. Though straightforward, this construction inherits vulnerabilities of the **hash's compression structure**.

2.1 Merkle-Damgard Hash Structure

Most legacy hashesfollow the Merkle-Damard design:

1. **Padding**: The message is padded to a multiple of block size (64 bytes) by appending 0x80, zero bytes, then a 64-bit length field.
2. **Compression Function**: Processes each 64-byte block sequentially, updating an internal state IV.
3. **Output**: The final state yields the hash digest.

Because the digest equals the final internal state, an attacker knowing H**(K || M) effectively knows the state after processing K || M.**

---

3. Length Extension Attack :

A length extension attack exploits the ability to resume hashing from a known internal state.

1. **Intercept** a valid (M, T) where T = H(K || M).
2. **Guess** the length of K, say L bytes.
3. **Compute** Padding for K || M as the hash would:
   - Let original bit length = 8*(L + |M|).
   - Padding = 0x80 + k bytes of 0x00 + 8-byte length.
4. **Resume Hashing:** Use the intercepted digest as the initial state and hash your extension E.
5. **Forge**: New tag T' = H_finalstate(E).
6. Construct the forged message:
   M_forged = M || padding || E
   and MAC **T'** verifies under **NaiveMAC**.

3.1 Numeric Example in MD5

- **Block size**: 64 bytes.
- **Suppose K length L = 16, |M| = 20.**
- **Total** = 36 bytes → needs 1 (0x80) + 27 (0x00) + 8 (length) = 36 padding.
- **Forge** E = "&admin=true".

The attacker computes:

new_mac, forged = hashpump(orig_mac, M, E, 16)

producing a valid **(M_forged, new_mac)** without knowing **K**.

---

4. Insecurity of NaiveMAC :

Because H(K || M) leaks internal hash state:

- **Incremental Processing**: Attackers forge MACs for extended messages.
- **Key Prepend Weakness**: Prepending doesn't bind the key to the outer hash structure.

Consequences: **Any system relying on this MAC is fully compromised—attackers can append arbitrary data and retain valid tags.**

C. Secure Alternative: HMAC

Defined in **RFC 2104**, HMAC wraps the hash in two layers:

$$HMAC(K, M) = H((K' \oplus opad) \| H((K' \oplus ipad) \| M))$$

- `K'` = `K` padded or truncated to block size.
- `ipad` = 0x36 repeated, `opad` = 0x5c repeated.
- Hiding of inner hash state under outer hash prevents length extension.

Security: Proven resistant to extension attacks and widely implemented in **cryptographic libraries**- (It is explained in more detail in the file (MAC and LEA (2.Mitigation Write-Up)

-*Thank You*-