# Demonstrating and Mitigating a Message Integrity Attack

Documented Verison

**Name** :

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

**COURSE** : Data Integrity and Authentication

**Doctor** : Maged Abdelaty

# A.What is a MAC and its purpose ?

A Message Authentication Code is a short piece of information used to authenticate a message and ensure its integrity and authenticity.

It is generated using a **secret key and a message, and then transmitted along with the message to the receiver**.

The receiver, who also knows the shared secret key, can **compute the MAC independently and verify that the message has not been altered and that it came from a trusted source.**

## Purposes of MAC :

1. **Data Integrity**: To ensure that the message has not been tampered with during transmission.

2. **Authentication**: To confirm that the message originates from a trusted party who possesses the secret key.

## B.How does a length extension attack work?

- A **Length Extension Attack** is a cryptographic attack that exploits the structure of certain hash functions such as **MD5** and **SHA1**, which follow the **Merkle-Damgard construction**.

  **MAC = hash(secret || message)**

- the attacker can exploit the hash function's behavior to **extend the message** and compute a valid MAC for the extended message **without knowing the secret key**.

## How the attack works:

1. The attacker intercepts a valid **(message, MAC) pair.**
2. They guess the **length of the secret key**.
3. Using the known MAC as the **internal state** of the hash function, the attacker **resumes hashing** and appends new data.
4. The attacker calculates the correct **padding** for **secret || message** as the hash function would do internally.
5. The attacker computes a valid MAC for **message || padding || new_data.**

This results in a **forged message** that appears valid to the server, despite the attacker **never knowing the secret key**.

## C. Why is MAC = hash(secret || message) insecure?

- These hash functions **process data in fixed-size blocks** and **update their internal state** as they go.
- Once an attacker knows the output of **hash(secret || message),** they effectively know the **internal state** of the hash function at that point.
- Using that state, they can **resume hashing** and **append arbitrary data**, crafting a new message with a valid MAC.

This vulnerability stems from the **Merkle–Damgard construction**, which makes hash functions **incremental** and thus prone to extension if the initial state is exposed.

## Consequences:

- Attackers can **forge valid messages** with appended malicious data.
- Integrity and authenticity are completely broken in systems relying on this MAC approach.

# D.Mitigation Write-up :

## a. Modifying the System to Use HMAC :

To prevent the length extension vulnerability, **we replaced the insecure MAC generation logic:**

**MAC = MD5(secret || message)**

with the secure and standardized **HMAC construction**:

**MAC = HMAC(secret, message)**

In Python, this is implemented using the built-in **hmac** library as follows:

```
import hmac, hashlib
MAC = hmac.new(secret, message, hashlib.sha256).hexdigest()
```

This modification was applied in the secure version of the server (**secureserver.py**) to enforce message integrity and authentication using an **approach that is resistant to cryptographic extension attacks**.

---

## b. Demonstration of Defense

After modifying the insecure **server.py** to a secure implementation in **secureserver.py**, we re-executed the same length extension attack from **client.py**.

Despite using the same forged message and forged MAC that bypassed the original MD5-based server, the secure HMAC-based server rejected the message as invalid.

This confirms that the attack, which was successful against a **naive MD5 MAC, fails completely when HMAC is used**. The security enhancement prevented the attacker from extending the message or forging a valid MAC without access to the secret key.

---

### c. Why HMAC Prevents Length Extension Attacks

**HMAC (Hash-based Message Authentication Code)** is specifically designed to be secure even when the underlying hash function (**SHA-256 or MD5**) is based on a vulnerable structure like the **Merkle–Damgard construction.**

HMAC uses a **two-layer hashing approach**:

**HMAC(K, M) = H((K ⊕ opad) || H((K ⊕ ipad) || M))**

Where:

- **K** is the secret key.
- **M** is the message.
- **opad** and **ipad** are fixed padding constants.

This design ensures that:

- The attacker cannot resume hashing from any internal state (**because the key is hashed twice, and in two different contexts**).
- The padding used internally in HMAC is controlled and cannot be predicted or simulated without the key.

---

### As a result:

- The length extension attack becomes ineffective.
- The MAC is secure even if the attacker knows **message** and **HMAC(secret, message)**.

---

### Conclusion

By replacing **hash(secret || message)** with the standard **HMAC(secret, message)** construction, we have successfully mitigated a critical vulnerability that allows attackers to forge valid message-MAC pairs. This change ensures both message integrity and authentication in a cryptographically secure way.

***-Thank You-***