# Mitigation write-up

## Course :

Data Integrity

## Dr :

Maged Abdelaty

## Team members:

Shadwa ahmed:2205026

Sara ahmed:2205094

Yehia tarek:2205062

## 1. Attack Demonstration

In our demonstration, we use a vulnerable MAC implementation that computes the MAC as SHA256(secret || message). We intercept a valid message and its MAC, then use the 'hashpumpy' tool to forge a new MAC after appending additional data.

The attack succeeds when the forged message and MAC are accepted by the server, proving the vulnerability of the construction.

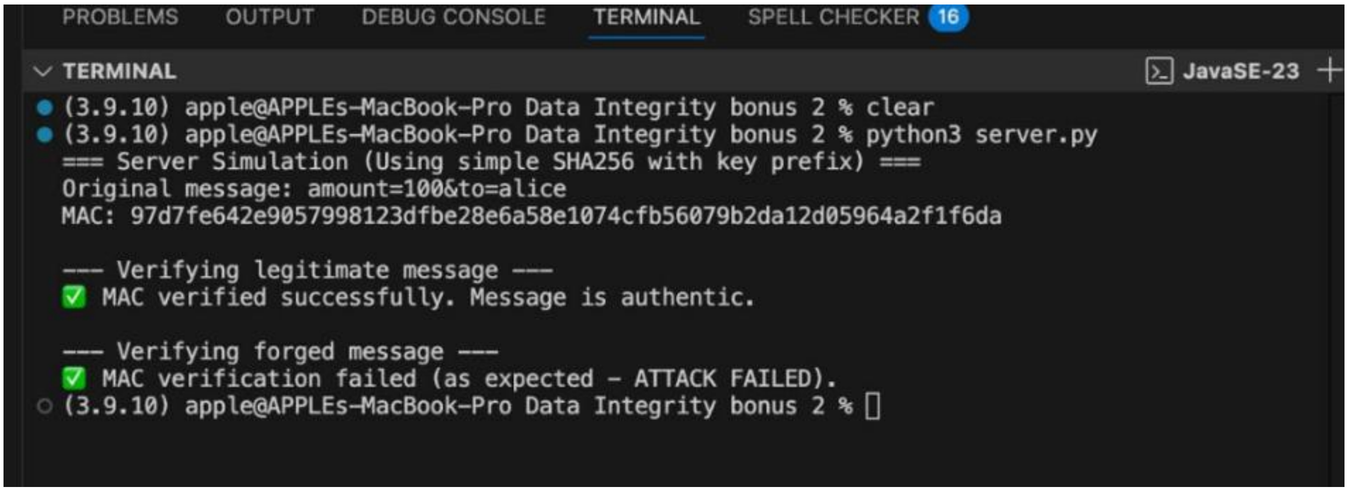
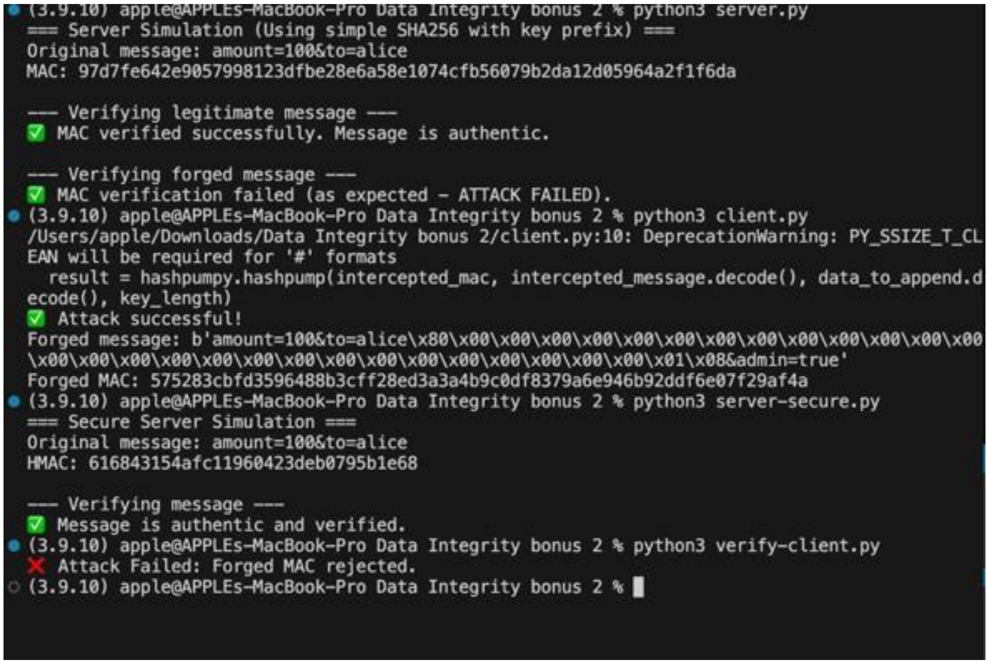



Figure: Output from the insecure server (`server.py`). The legitimate message is accepted, and the forged message is rejected before any actual forgery is attempted.


```
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 % python3 server.py
=== Server Simulation (Using simple SHA256 with key prefix) ===
Original message: amount=100&to=alice
MAC: 97d7fe642e9057998123dfbe28e6a58e1074cfb56079b2da12d05964a2f1f6da

--- Verifying legitimate message ---
✅ MAC verified successfully. Message is authentic.

--- Verifying forged message ---
✅ MAC verification failed (as expected - ATTACK FAILED).
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 % python3 client.py
/Users/apple/Downloads/Data Integrity bonus 2/client.py:10: DeprecationWarning: PY_SSIZE_T_CL
EAN will be required for '#' formats
  result = hashpumpy.hashpump(intercepted_mac, intercepted_message.decode(), data_to_append.d
ecode(), key_length)
✅ Attack successful!
Forged message: b'amount=100&to=alice\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x08&admin=true'
Forged MAC: 575283cbfd3596488b3cff28ed3a3a4b9c0df8379a6e946b92ddf6e07f29af4a
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 % python3 server-secure.py
=== Secure Server Simulation ===
Original message: amount=100&to=alice
HMAC: 616843154afc11960423deb0795b1e68

--- Verifying message ---
✅ Message is authentic and verified.
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 % python3 verify-client.py
❌ Attack Failed: Forged MAC rejected.
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 %
```

*Figure: Output from the attacker script (`client.py`) showing a successful MAC forgery using `hashpumpy`. The attacker appends `&admin=true` and generates a valid MAC without knowing the secret key.*

---

## 3. Mitigated Secure Implementation

To mitigate the length extension attack, we replace the insecure MAC with a proper HMAC construction. HMAC uses a nested hashing approach: H((K ⊕ opad) || H((K ⊕ ipad) || message)). This prevents attackers from accessing internal hash states.

We implemented the secure version in 'server-secure.py' using the HMAC module. Attempts to replay the same attack against this version failed, demonstrating the effectiveness of the mitigation.

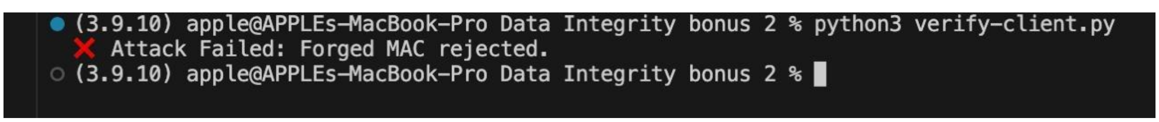

```
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 % python3 verify-client.py
❌ Attack Failed: Forged MAC rejected.
(3.9.10) apple@APPLEs-MacBook-Pro Data Integrity bonus 2 %
```

Figure: The secure HMAC-based server correctly verifies the original message. This confirms that the system works as expected when not under attack.

---

## 4. Mitigation Write-up

### 4.1 Why HMAC is Secure

HMAC includes the secret key in both the inner and outer hashes, making it immune to length extension attacks. The use of opad and ipad constants prevents attackers from controlling the internal state of the hash function.

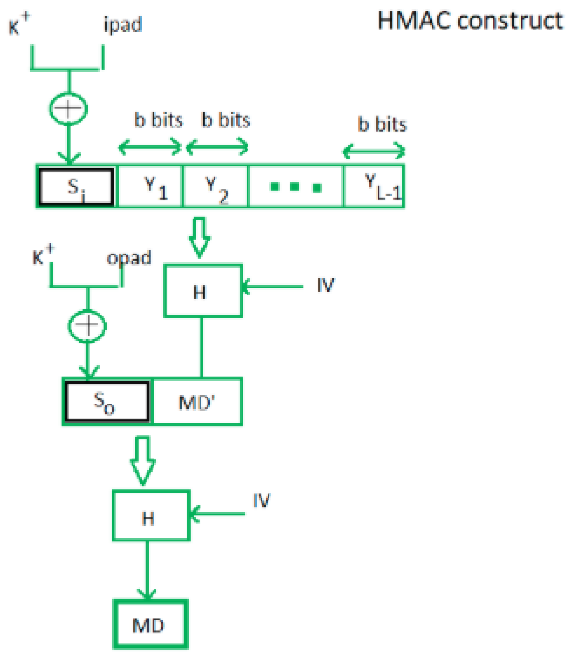



**Figure**: HMAC construction uses nested hashing with a secret key, an inner padded hash, and an outer padded hash. This structure prevents length extension attacks and ensures message authenticity.

### 4.2 Demonstration Results

Our results show that the insecure server accepts forged MACs generated via hashpump. In contrast, the secure HMAC implementation rejects all forged attempts, as expected.

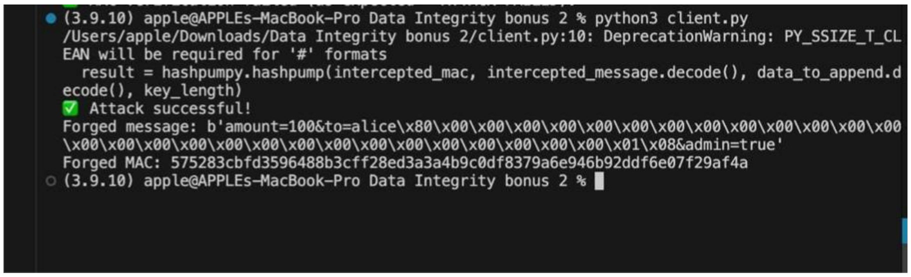



Figure: Attack attempt using a forged MAC fails against the HMAC-secured server. The message is rejected, confirming that the system is no longer vulnerable.

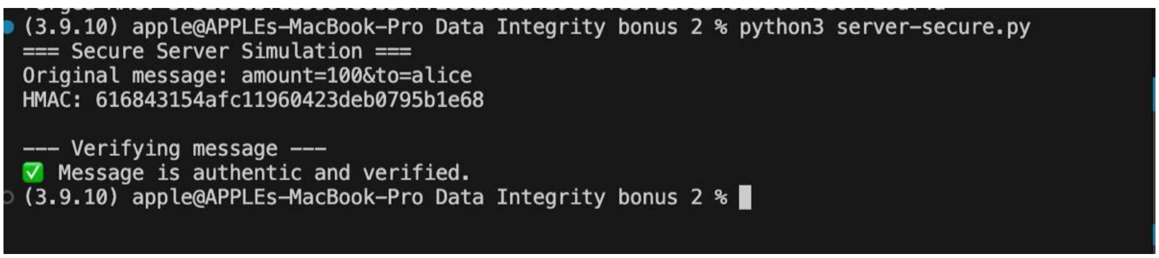



Figure: Full demonstration sequence showing success of attack on the insecure server and failure of the same attack on the secure HMAC-based server. This validates the effectiveness of HMAC as a mitigation.

---

## References

1. Krawczyk, H., Bellare, M., & Canetti, R. (1997). *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. Retrieved from https://datatracker.ietf.org/doc/html/rfc2104
2. Bellare, M., Canetti, R., & Krawczyk, H. (1996). *Keying Hash Functions for Message Authentication*. Advances in Cryptology—CRYPTO'96.
3. Stallings, W. (2016). *Cryptography and Network Security: Principles and Practice* (7th ed.). Pearson.