Mitigation Writeup

Team Members

Hossam Ahmed Aldesouky	**2205097**
Omar Hossam Ahmed	**2205150**
Tarek Gamal Elkelany	**2205029**

Proper Mitigation – Using HMAC

- **Secure Construction**:

```
import hmac
def generate_mac(message: bytes) -> str:
    return hmac.new(SECRET_KEY, message, hashlib.md5).hexdigest()
```

· **Why HMAC is Secure**:

- Inner and outer key padding.
- Defends against length extension
- Cryptographic security proof under certain assumptions.

Verifying Attack Failure on HMAC

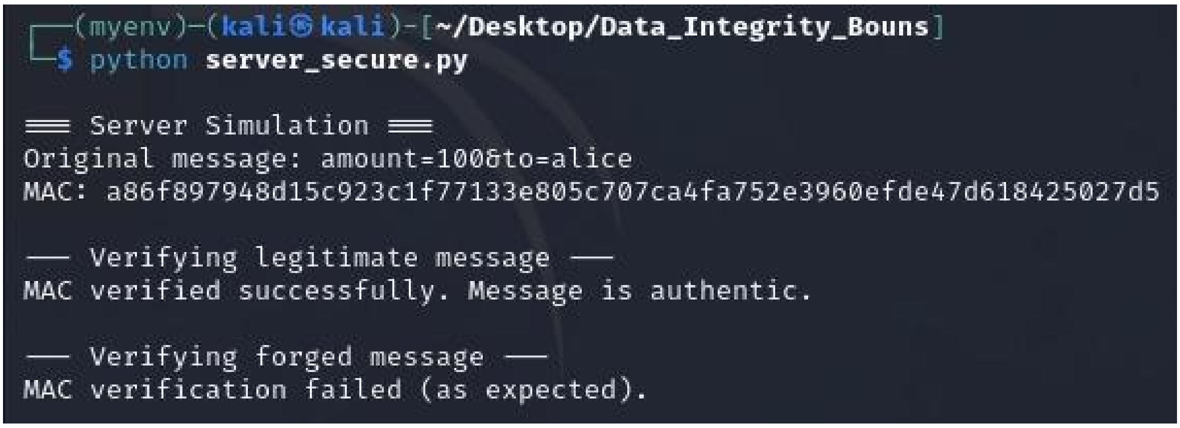
- **Re-run attack against updated server**

- **Result**:

  - Forged message is rejected
    - MAC mismatch (attacker cannot generate valid HMAC without secret)

Screenshot:

Show failed verification on server output



Conclusion

- Naive MAC constructions like MD5(secret || message) are **vulnerable** to length extension and brute-force attacks, allowing attackers to forge valid MACs without knowing the secret.

- Length extension attacks exploit the internal workings of hash functions (e.g., MD5, SHA1), enabling message tampering with valid MACs.

- Brute force can reveal weak secret keys if predictable or short keys are used, further compromising message integrity.

- **HMAC** provides a secure alternative by using inner and outer key padding, preventing length extension and resisting brute-force forgery.

- Implementing **HMAC** and using strong, unpredictable keys effectively **mitigates these attacks**, ensuring message authenticity and integrity.

- Demonstrations confirmed:
  - Attacks succeed against vulnerable MACs.
  - Attacks fail when HMAC is properly implemented.